NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A DATA ORIENTED APPROACH TO
INTEGRATING MANUFACTURING FUNCTIONS IN
FLEXIBLE MANUFACTURING SYSTEMS

by

David R. Fleischman

June 1988

Thesis Advisor: C. Thomas Wu

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (*If applicable*) Code 52	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, California 93943-5000	7b ADDRESS (*City, State, and ZIP Code*) Monterey, California 93943-5000
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (*If applicable*)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (*Include Security Classification*)
A Data Oriented Approach to Integrating Manufacturing Functions in Flexible Manufacturing Systems

12. PERSONAL AUTHOR(S)
Fleischman, David R.

13a. TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (*Year, Month, Day*)	15 PAGE COUNT
Masters Thesis	FROM _____ TO _____	1988 June	88

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

17 COSATI CODES			18 SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	Computer Integrated Manufacturing; Flexible Manufacturing Systems; data modeling

19 ABSTRACT (*Continue on reverse if necessary and identify by block number*)

Computer Integrated Manufacturing (CIM) seeks to integrate computers into the manufacturing environment, with the end result being a more efficient and productive factory. Current approaches to CIM generally fail to truly integrate the various manufacturing functions (design, scheduling, planning, manufacture, business, etc.) and instead result in self-sufficient, computer-served "islands of automation." In these systems, data must be translated before moving from one manufacturing function to another.

Wu and Madison have approached data modeling in a CIM environment from a new perspective. Their approach seeks to provide one data model that meets the needs of all manufacturing functions within a factory, negating the need for human or machine data translators.

In this thesis, we review the work done by Wu and Madison and apply their data model to a particular manufacturing function, the Flexible Manufacturing System (FMS).

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Professor C. Thomas Wu	22b TELEPHONE (*Include Area Code*) (408) 646-3391 22c OFFICE SYMBOL Code 52Wq

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆ U.S. Government Printing Office 1986—606-24-

Approved for public release; distribution is unlimited.

A Data Oriented Approach to Integrating Manufacturing Functions in Flexible Manufacturing Systems

by

David R. Fleischman
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1976

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

June 1988

ABSTRACT

Computer Integrated Manufacturing (CIM) seeks to integrate computers into the manufacturing environment, with the end-result being a more efficient and productive factory. Current approaches to CIM generally fail to truly integrate the various manufacturing functions (design, scheduling, planning, manufacture, business, etc.) and instead result in self-sufficient, computer-served *islands of automation*. In these systems, data must be translated before it moves from one manufacturing function to another.

Wu and Madison have approached data modeling in a CIM environment from a new perspective. Their approach seeks to provide one data model that meets the needs of all manufacturing functions within a factory, negating the need for human or machine data translators.

In this thesis, we review the work done by Wu and Madison and apply their data model to a particular manufacturing function, the Flexible Manufacturing System (FMS).

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. OVERVIEW

As the computer moved from the laboratory of the scientist into the office of the engineer and, finally, into the factory, manufacturers had high expectations. A new technology, which came to be known as Computer Integrated Manufacturing (CIM), held great promise in terms of cutting costs, increasing efficiency, and raising profits. It appeared to be a simple problem on the surface, integrating computers into the factory. Once installed, computers, the purported "cure-all" for industry, would solve all the problems associated with design, production, scheduling, and analysis and forecasting of market conditions. What held great promise a decade ago, though, still has yet to be fully implemented in the factory. This thesis examines why this is so, offers a potential solution, and implements the proposed solution in a Flexible Manufacturing System (FMS) environment.

B. HISTORICAL BACKGROUND

Within the last fifteen years, the manufacturing industry has undergone major changes in an effort to keep pace with an ever-changing marketplace. What had been a personal relationship between designer and machinist rapidly evolved into a complex process utilizing specially-trained personnel, sophisticated data and communications handling equipment, and advanced machine tools [Ref. 1]. This evolution naturally led to a fragmentation of the design and production processes as the need for specialized people to handle specialized tasks became apparent. With this fragmentation, in turn, was lost much of the gain in efficiency and productivity which specialization brought. Fragmentation meant increased overhead as more people and more equipment were

required to manage and control the necessary systems to move information and materials. Computers were introduced in small parts of companies to assist in this management and control function, further fragmenting the whole process. The end result was highly-specialized "cells" within a company composed of people and machines tasked to perform one or a small number of missions. Few, if any, people had an understanding of the entire manufacturing process from design to finished product.

Overall, the improvements in efficiency and productivity were not as great as had been expected due, in large part, to the approach to the problem. Rather than viewing the application of automation and computers in terms of the overall manufacturing process, a very narrow approach was used, applying automation only to the individual phases of the overall process.

Computers were also introduced into other facets of manufacturing. Dynamically programmable machines, first numerical control (NC) machines using paper-tapes, and later, computer numerical control (CNC) machines, streamlined the various manufacturing processes. Data management systems replaced manual inventory control and accounting systems, thus increasing the ability of management to control the growing business. Computer Aided Design (CAD), the process of computer-assisting the human engineer to convert his ideas and knowledge into mathematical and graphical models, greatly expedited the design process. The net effect of this automation effort was a significant increase in productivity, efficiency, cost savings, quality; all factors vital to the flourishing of an enterprise. Unfortunately, this effort carried with it the unwanted side effect of an uncontrolled, fragmented management information system.

As the factory and the front-office were automated, the need for interoperability was often ignored, resulting in the inability to communicate or exchange data among the various entities within the business. Design could not pass information to

Manufacturing, Manufacturing received orders manually from Customer Service, and Shipping hand-carried invoices to Customer Service; incompatability and fragmentation were rampant. Each department was automated and internally efficient, yet external interactions were slow and inefficient. The need to integrate these islands of automation; isolated activities within the larger enterprise, each employing incompatible automation techniques, became apparent.

C. THE NEED FOR COMPUTER INTEGRATED MANUFACTURING

Initially, it was the manufacturing people, the mechanical and industrial engineers, who expressed the need to make these diverse computer systems communicate with each other. They foresaw a system which allowed, for instance, design data for a product from a Computer-Aided Design (CAD) system to be used directly by a Computer-Aided Manufacturing (CAM) system for process planning. The first approach to providing integrated, computer-supported manufacturing was the development of a translator, or interface, to convert data from the format used by one computer into another format required by a different computer. The advantage of this approach is that it does not require any modifications to existing systems, however, it really does not integrate the various phases of the overall process, it merely connects them. Additionally, it introduces the problem of software maintenance; as any component in the system is modified, the translator must be rewritten or modified. This approach has proven effective as a short-term solution, but does not provide the answer to the need for true integration [Ref. 2].

A second approach has been developed by the Society of Manufacturing Engineers (SME), the CIM Enterprise Wheel. This approach features a centralized data management system and, on the surface, appears to meet the need for a method of providing data across the entire spectrum of manufacturing phases. However,

implementation is difficult with current database management systems. According to some within the database community, the relational database management system is not able to handle the complexities of the manufacturing environment2 . It is the opinion of these people that either an improved relational system must be developed, or a completely new approach to serving data be investigated.

This new approach involves looking at the centralized data server not as a physical entity but as a logical one; actually a common data model meeting the needs of all phases of the manufacturing process. This approach, with one major modification, is the one which shows the greatest potential for truly integrating all aspects of manufacturing. If true integration is the goal, then it is the model derived from this approach which holds great promise for achieving that goal. We intend to attack the problem of true integration, in the context of FMS, in this thesis.

This thesis is organized as follows: Chapter 2 provides several definitions of CIM, as well as a review of several data models currently used to support CIM and the shortcomings of these models; Chapter 3 reviews the an innovative data-oriented approach to CIM, a method which looks at the problem of integration from a data-oriented perspective rather than the traditional process-oriented point of view; Chapter 4 describes FMS; and in Chapter 5, we apply the Manufacturing Data Model to a FMS application and make observations regarding the fitness of this model for accomplishing the integration in CIM.

II. CURRENT APPROACHES TO CIM

A. INTRODUCTION

In this chapter, we introduce a number of definitions of Computer Integrated Manufacturing (CIM), discuss three current approaches to CIM, describe the data models used to support these approaches and the modeling abstractions which apply, and conclude with a discussion of the shortcomings of these approaches.

B. COMPUTER INTEGRATED MANUFACTURING DEFINED

There are almost as many definitions of CIM as there are papers and texts on the subject. The following definitions cover the spectrum of what is generally regarded as CIM:

* CIM is a series of interrelated activities and operations involving the design, material selection, planning, production, quality assurance, management, and marketing of discrete consumer and durable goods [Ref. 3];

* CIM is a network of computer systems integrating the various manufacturing processes [Ref. 1];

* CIM is the deliberate integration of automated systems into processes resulting in the production of a product [Ref. 1];

* CIM is the logical organization of individual engineering, production, and marketing/support functions into a computer integrated system [Ref. 1];

* CIM is the phased implementation of automated and non-automated systems to support the manufacturing environment [Ref. 1];

* CIM is an information structure providing a flow of data needed by the various functions in the manufacturing process [Ref. 1];

* CIM is a strategy, incorporating computers, to link existing technology and people to optimize business activity [Ref. 4].

While some definitions summarize the objectives and meaning of CIM better than others, all support the overall goal of CIM - to complete production of an end-item in the

simplest and most timely fashion, with a minimum of human intervention, with a minimum of interruptions in the flow of all required processes and at minimum cost. Implementation of CIM should mean real-time shared access to all data by those people and processes requiring access, as well as better quality products, shorter response time to design changes, shorter design and production times, and more efficient processing of small orders [Ref. 1].

C. THE PROCESS OF INTEGRATION

Traditional processes performed within a factory include design, process planning, Numerical Control (NC) machine programming, robot programming, quality control, testing, shop floor management, marketing, sales estimating, order processing, scheduling, material requirements planning, plant maintenance, shipping, inventory management, purchasing and accounting [Ref. 1]. These processes, in turn, have been grouped to form manufacturing functions, such as Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), Computer Aided Process Planning (CAPP), Computer Aided Test (CAT), Group Technology (GT), and Flexible Manufacturing Systems (FMS). These functions are not always grouped consistently; for example, in one case robot programming might be considered a process in CAM, while in another case it may be included as part of CAPP. This ambiguity ("Which processes constitute which manufacturing functions?") is a significant problem faced by systems designers seeking to integrate computers into the factory and is an area in need of standardization.

D. CURRENT APPROACHES TO INTEGRATION

There are three approaches which are currently applied to integrating manufacturing functions. Each approach requires a data model or models to pass information between the processes and functions and each of these models are supported by one or more modeling abstractions. A discussion of these approaches follows.

1. The High Level Approach

The high level approach to integration seeks to utilize as much of the existing manufacturing machinery and automation equipment as possible, thereby holding down the investment in dollars and minimizing time required for total integration. This approach is commonly used to provide interfaces for the four main components of CIM, as shown in Figure 1 [Ref. 1]. The typical method of implementing the high level approach is through a translator, a software system which takes output data from one process and converts it into a form which can be used as an input to another process.

This approach has several drawbacks: when the data format used in one process is modified, the translator must be rewritten, translation is costly in terms of execution time and data which is passed from one process to another and then back again requires two translations. In general, the high level approach does not *solve* the integration problem but instead provides a "work-around." For these reasons, the high level approach should be considered a short-term solution.

2. The Centralized Database Approach

The second approach to integration involves interfacing the four main components of CIM through a centralized database, as shown in Figure 2. This approach, while desirable from the standpoint of query processing and database maintenance, is extremely difficult to implement. It requires a database management system which can support the multitude of functions requiring data as well as assign priorities to multiple requests for real-time access. The difficulty in implementing a database management system which can provide these functions makes the centralized database approach a long-term solution to integration.

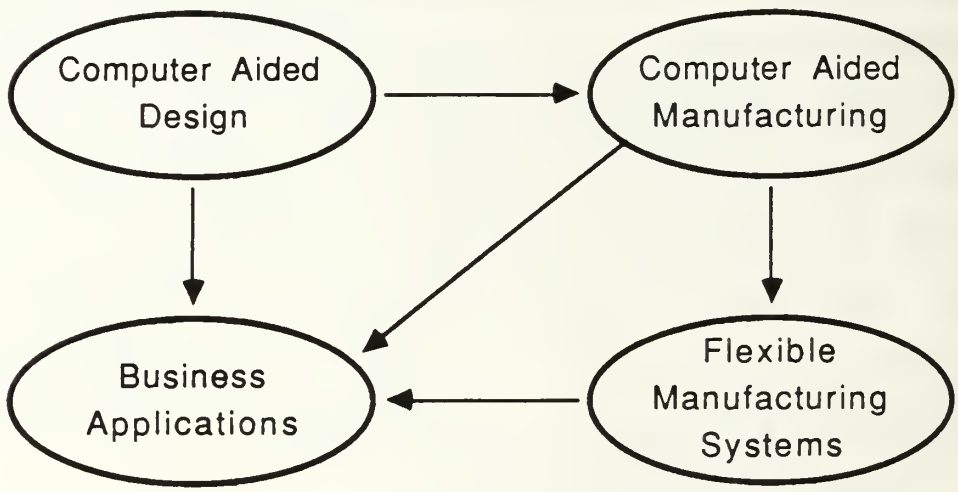


Figure 1. High Level Integration

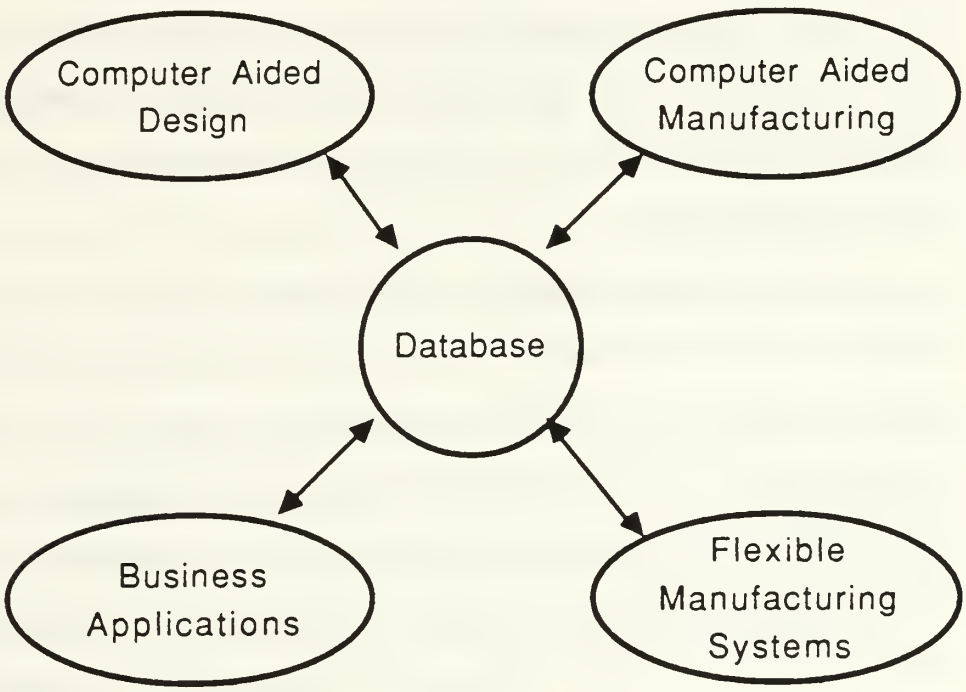


Figure 2. Centralized Database Approach

3. The Low Level Approach

The third approach to integration breaks the overall manufacturing process down into manufacturing functions, such as those discussed in section C above. These in turn, are served by individual databases. Interfaces would be provided between databases and manufacturing functions, enabling one function to use data from another in a network-like arrangement. Such an interface would require that all the function databases use a common language and data model. Finding such a language and data model, both of which must be powerful and flexible enough to support the variety of abstraction principles utilized in each of the manufacturing functions, is a major obstacle to this approach [Ref. 1].

One advantage of this approach is that the use of a uniform model and language would force standardization among CIM system manufacturers, bringing with it several advantages such as ease of maintenance and upgrade and a yardstick to judge competing systems. The low level approach also has the advantages of all distributed database systems: they are reliable, available and fast. But with these advantages come some disadvantages. Distributed databases are expensive to develop, prone to delay-causing bugs and have inherent processing overhead [Ref. 5]. An additional advantage of the low level approach is the more manageable data volume associated with any one database. Consider the centralized approach, with its huge database; maintenance and modification become a major undertaking. The low level approach, again, is considered a long-term solution, requiring careful planning and integration.

4. Selection of an Approach

Past research has demonstrated that, while each of the three approaches has the potential for application, only the high and low level approaches exhibit the characteristics considered necessary to truly integrate manufacturing functions. The centralized database approach would require a computer capable of monitoring the entire

manufacturing process as well as maintaining the database, including both static data (setpoints, alarms, unit conversions, etc.) and dynamic data (dimensions, locations, current system values and states, etc.), controlling operator access, and a multitude of other functions [Ref. 6]. This extensive list of responsibilities would overwhelm all but the most capable and expensive of those computers currently available. Other problems exist with this approach. There is the possibility of a single-point failure causing a total system shutdown, if the single-point happens to be the host computer. The solution to this potential problem is the purchase, at some large sum, of a backup computer. Additionally, the massive size of the database, in terms of data communications requirements, storage, and manipulation, exceed the capabilities of today's database management systems and would tax a modern mainframe computer. This is simply not a viable approach.

The two remaining approaches, high and low level integration are both viable. Both have been developed to some extent. However, the low level approach shows the most promise for solving the problems facing the designer of truly integrated manufacturing systems. In this approach, the data model serves a critical function.

E. THE DATA MODEL

As discussed above, the low level approach to integration uses distributed databases, one for each manufacturing function, to support the CIM concept. These databases must be linked by a common data model and a uniform language, each powerful enough to support the different semantics inherent in the various manufacturing functions. The idea of a powerful, flexible language is well understood and documented and we do not intend to develop that concept in this thesis. The question of a data model remains - is there a single model which could handle the diverse data needs of all the manufacturing functions? We will examine current techniques in data modeling in support of CIM.

The data model is the group of general rules for the specification of the structure of the data, along with the operations allowed on the data. In other words, a data model is an abstraction, or group of abstractions, which allows us to see the forest (information content of the data) rather than the trees (individual values of data) [Ref. 7]. To better understand the concept of data modeling, it is useful to define the objects to be modeled.

A working definition of an atomic piece of data might be the following tuple:

<object name, object property, property value, time>.

This is a reasonable way to view an idea; the tuple represents an object (object name) and an aspect of the object (object property) which is instantiated by a specific value (property value) at a particular time (time) [Ref. 8]. Time is typically a very cumbersome aspect of data modeling, not necessarily appropriate, and often difficult to encode. As a result, time will not be considered in this thesis. The definition of an elementary datum reduces to *<object name, object property, property value>*. Several data modeling methods have been developed to represent and relate these three remaining elements of the datum. One typical method is to put data into categories according to properties [Ref. 9]. In this method, the names of the categories, together with their properties, is known as a *schema*. The schema can also include relationship data between the categories and their properties. Figure 3 is an example of a schema with three categories, **US Government, Commanding Officer** and **USS LaJolla.** The categories are represented by ovals, the properties by rectangles, and the relationships are shown as lines connecting the categories they relate to.

The data structure used to represent the categories, combined with the set of allowable operations on those data structures, defines a particular data model. The number of possible combinations of structures and operations would indicate that a

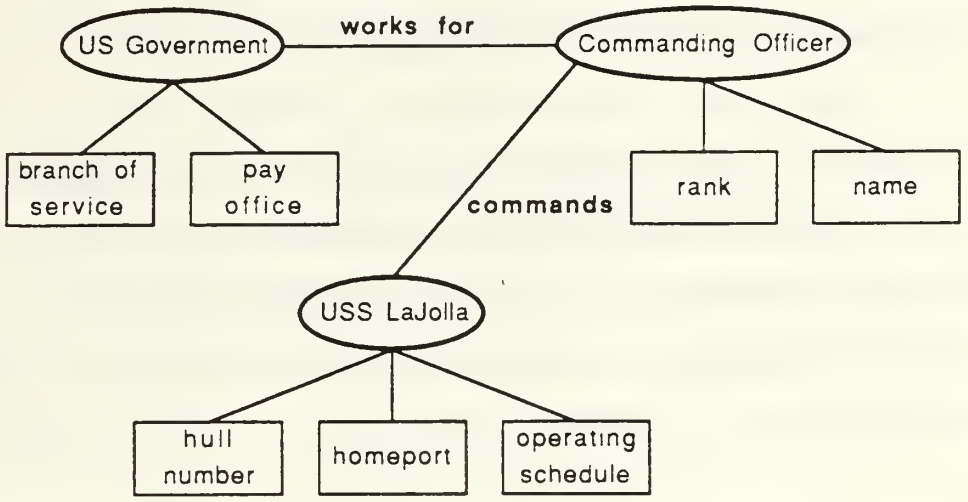


Figure 3. Example of Database Schema

significant number of data models could be specified; however, in reality only a limited number of models have practical uses. Of these, three, the hierarchical, network, and relational models, are widely used and accepted. These models, also known as *classical* data models [Ref. 10], will be discussed in the following paragraphs.

1. Classical Data Models

a. Hierarchical

The hierarchical data model, the oldest of the traditional data models, is a direct extension of the hierarchical database concept. The hierarchical data model represents objects and properties as nodes in a tree, with the relative order of trees and subtrees important in defining the relationship between nodes. The arcs connecting the nodes point away from the root and toward the leaves of the tree. Figure 4 shows an

intension of a hierarchical database for a submarine. The database is depicted in terms of its nodes, or *segment types*, and the relationships between them.

In Figure 4, **submarine**, **ship's attributes** and **crew's attributes** represent segments, with the segments further broken down into one or more data items or *fields*. The relationships in a hierarchical data model are called *parent-child relationships*, and can be one-to-one or one-to-many [Ref. 7]. In Figure 4, the relationship between **submarine** and **crew's attributes** is one-to-one, that is, the **submarine** has one set of **crew's attributes**. This relationship is represented by a single arrow pointing at **crew's attributes**. Conversely, several sets of **ship's attributes** could be applied to the node, **submarine**, depending on the class. This one-to-many relationship is represented by

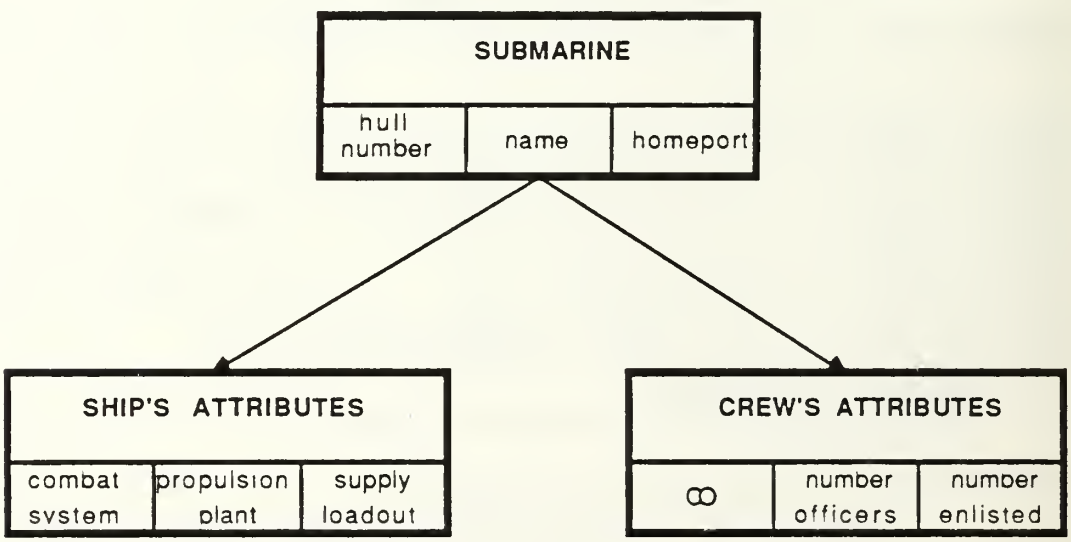


Figure 4. Intension of a Hierarchical Database

double arrows pointing at **ship's attributes.** In this example, **submarine** is the parent of both **ship's attributes** and **crew's attributes; ship's attributes** and **crew's attributes** are *siblings.*

Figure 5 depicts a record, an *extension* of the structure shown in Figure 4. An extension of a segment is a group of data items relating to one particular entity [Ref. 6].

The hierarchical data model, while appealing because of its simplicity, has two inherent constraints: all relationships must be binary, either one-to-one or one-to-many, and all relationships must be capable of representation in a tree-like depiction [Ref. 7]. The constraint that all relationships be binary means that this model can only represent one-to-one and one-to-many relationships. It is not possible to

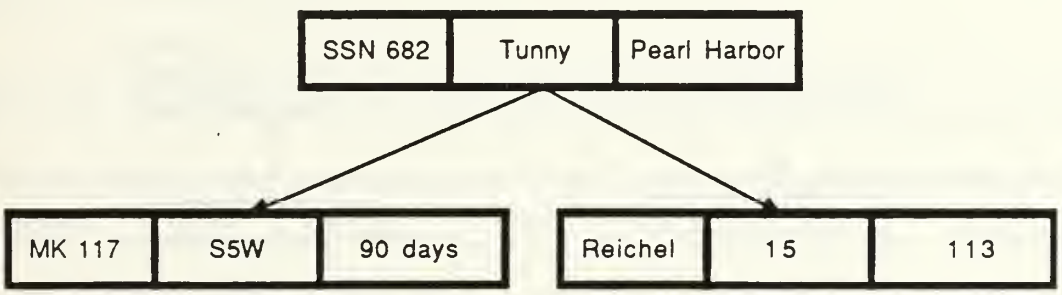


Figure 5. Extension of a Hierarchical Database

represent many-to-many relationships directly; instead, one of two artificialities must be introduced. Figure 6 shows one method, known as *duplication*. In this method, duplication of **job number** records are used to indicate that many **ships** are repaired by many **shipyards.**

The second technique requires the introduction of second tree, as shown in Figure 7. Here again, **ship** has a many-to-many relationship with **shipyard.** Both of these methods, by retaining superfluous records, the potential for data duplication.

The second inherent constraint of hierarchical models is the requirement that all data relationships be represented by a tree structure. As long as the data are naturally hierarchical, this constraint does not present a problem. However, many-to-many and multiple parent (many-to-one) relationships require modification to the graph

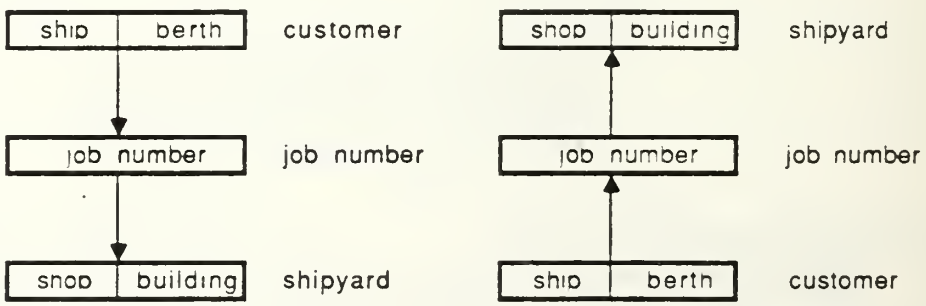


Figure 6. Many-to-Many Relationship Represented by Duplication

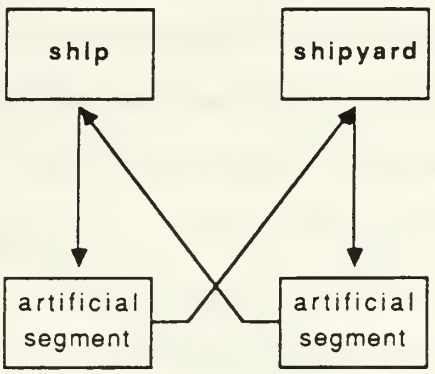


Figure 7. Many-to-Many Relationship Represented by Artificial Segment

structure. As a result, these types of relationships are not capable of being represented by hierarchical data models. and are not capable of being represented by hierarchical data models.

b. Network

Network data models are based on tables and graphs. The most prominent network data model to date is the model developed by the Data Base Task Group of the Conference on Data Base Systems Languages, known as the CODASYL model [Ref. 11]. In this model, the nodes of the graph represent *record types* which correspond to groups of related fields. The lines between nodes correspond to *set types* which represent the connections between tables. Each connection, or *set*, has a designated *owner* record type and may contain zero or more *member* record types. Figure 8 shows an example of a network data model.

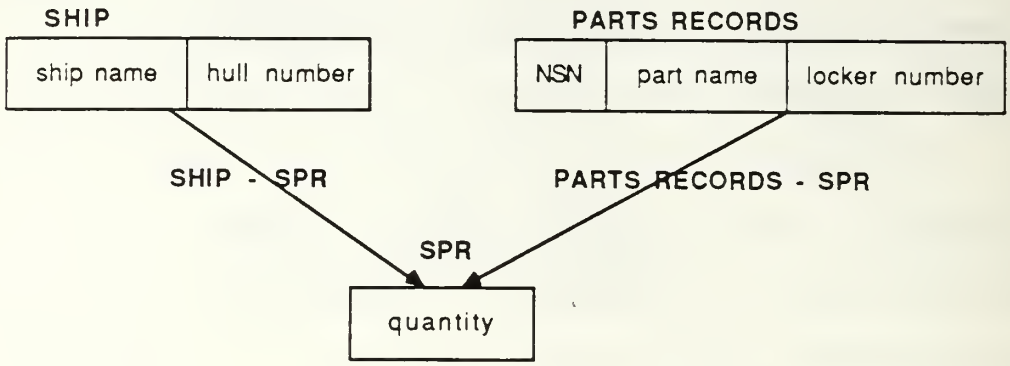


Figure 8. Example of a Network Database

In this example, there are three record types: **ship, parts records** and **SPR.** The sets **ship-SPR** and **parts records-SPR** respectively, relate **ship** and **parts records** to **SPR.** Each occurrence of **ship-SPR** consists of one record from **ship** (the owner) and one record from **SPR** (the members). In a similar fashion, an occurrence of **parts records-SPR** would consist of a single record from **parts records** (the owner) and one record from **SPR** (the members).

One constraint of the network data model is that, as in the hierarchical model, all relationships must be functional, one-to-one or one-to-many. Additionally, network data models cannot be used to represent recursive relationships, ie., the situation where both owner and member record types are the same [Ref. 7].

c. Relational

The relational data model is significantly different from the hierarchical and network data models, both in its basis and its approach. The model is based in relational mathematics, a field centered around the concept that a relation can be defined which expresses the correspondence between two sets. Its approach is more abstract than the hierarchical or network models and, therefore, more natural in its representation of data. With the hierarchical and network models, data is forced into an artificial construct, either a hierarchy or a set. In most cases, these models tend to complicate the user's view of the data. The relational data model's strongest point is that it simplifies, rather than complicates the representation of data. [Ref. 6]

Without delving deeply into the mathematics of the relational model, we will provide a brief overview of the concept involved. The model is based on the notion of a *relation,* or expression relating two or more sets. A relation can be thought of as the Cartesian product of the domains of the sets involved. In even simpler terms, a relation is the tuple which results when you take the Cartesian product of the domains of those sets.

Typically, relations are represented to the user as a two-dimensional table where the column headings represent *attributes* and the rows represent the *tuples. Figure* 9 depicts a relation called **SUBMARINE.** This relation has four attributes, **hull number, name, Commanding Officer** and **homeport.** The domain of the attributes are character strings of length 3, 15, 15 and 15, respectively.

Tuples are identified by the values of their attributes. For example, **682, TUNNY, Kaup, Pearl** uniquely identifies the first tuple in Figure 9. Often, however, it is not necessary to use all the attributes of a tuple to uniquely identify it. The attribute **San Diego** does not uniquely identify a tuple (three tuples contain that attribute) but the

SUBMARINE

hull number	name	Commanding Officer	homeport
682	TUNNY	Kaup	Pearl
701	LaJOLLA	MacNeill	San Diego
707	PORTSMOUTH	Ihrig	San Diego
713	HOUSTON	Rogers	San Diego

Figure 9. Example of Relation

pair of attributes, **707** and **San Diego** does uniquely identify a tuple. A combination of attributes which uniquely identifies a tuple is called a *candidate key*. If a candidate key is chosen to be used as the tuple identifier, it is referred to as the *primary key [Ref. 5]*.

Relational models exhibit *data independence*, a measure of a database system's ability to allow for change in the database without necessitating change in the database programs or application programs. The model achieves this by representing data as relations and then manipulating the relationship between relations through relational calculus or relational algebra [Ref. 5]. Additionally, the relational model represents data logically rather than requiring it to fit into an unnatural construct as hierarchical and network models do.

d. Shortcomings of Classical Data Models

The classical data models have some significant shortcomings, as documented in [Ref. 12]. Two of these shortcomings are particularly applicable to the CIM

environment [Ref. 6]. These two limitations are a lack of support for abstract data types and limited semantic expressiveness. The more serious of these is the problem of limited semantic expressiveness [Ref. 6]. Simple data structures in the classical models often cause loss of information and minimal support for modeling of application environment semantics [Ref. 13]. These models cannot distinguish between the different types of relationships between objects, in fact, the same data structures used to model attributes of an object are used to model the type of the object and the relationships between objects. The result is loss of data [Ref. 6].

The second problem, the lack of support for abstract data types, causes complex objects from the application environment to be represented by record data structures. This unnatural representation requires users to address and manipulate objects from the application environment differently than they would be addressed and manipulated in the data modeling environment. This directly counteracts the primary purpose of data modeling. [Ref. 6]

2. Higher-level Data Models

a. Introduction

Considerable effort is currently being directed at higher-level data models which provide greater flexibility and expressiveness than the three traditional models. These models, also known as *semantic* data models, seek to achieve increased database accessibility by end users. To achieve this objective, the semantic models embed the semantics appropriate to the application [Ref. 6].

Semantic data models employ *abstraction concepts*, or ideas, to organize the information they represent and hide detail thereby reduce complexity. The following section discusses the most common abstraction concepts used in semantic data models.

b. Abstraction Concepts

(1) Generalization/Specialization. *Generalization* views a set of tokens or a set of types as one generic object [Ref. 7]. By generalizing, we can overlook many of the individual differences between objects, emphasizing instead their similarities. Figure 10 depicts a generalization hierarchy for a submarine crew. The arrows indicate the direction of generalization. In this example, **crew member** is a generalization of **officer** and **enlisted.** One advantage of generalization is that the idea of *inheritance*

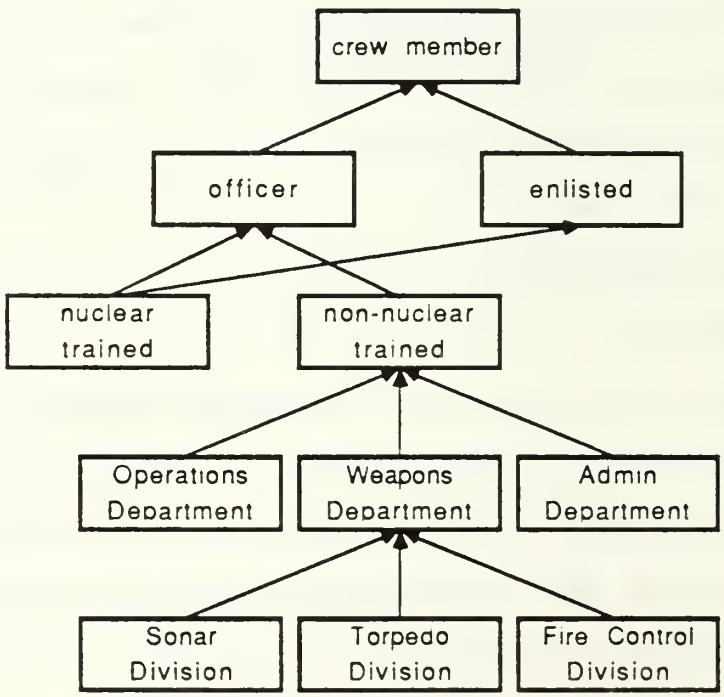


Figure 10. Generalization and Specialization

applies, that is, the properties of the generalized type are also properties of those entities further down in the hierarchy. For example, in Figure 10, all of the properties of **crew member** are inherited by **enlisted** (assigned to a submarine, submarine-trained, etc.), properties which are inherited further down by various classifications of **enlisted.** Inheritance of certain properties can be disallowed while some properties can be specified as applicable to only one type.

Specialization is the opposite of generalization [Ref. 7]. In Figure 10, **nuclear trained** is a specialization of **enlisted.** In specialization, inheritance does not always apply. In our example, all **Weapons Department** personnel are **non-nuclear trained** but it is not true that all **non-nuclear trained** personnel are in the **Weapons Department.**

(2) Aggregation. *Aggregation* is the abstraction concept in which an object is represented by its constituent parts and the relationships between those parts [Ref. 14]. This concept is useful because it makes visible both the structure of an object and the individual components of the object and how they relate to the structure of the object and to each other [Ref. 7]. Built into this abstraction concept is the ability to hide from the user those details of implementation which he does not need to know. Aggregated properties which are definitional in nature are called *intensional* properties [Ref. 15]. The values that these intensional properties can take on are referred to as *extensional* properties [Ref. 15]. Extensional properties are factual. Figure 11 depicts an aggregation hierarchy. Primitive objects, objects which can not be further subdivided, are displayed in lower case. Aggregated objects are shown in upper case. In this example, **name, hull number** and **homeport** are primitive objects while **SYSTEMS** and **CREW** are aggregated objects. **Name, hull number, homeport, SYSTEMS** and **CREW** are intensional properties. "LaJolla", "701", "San Diego" "propulsion", "combat", "navigation" and "Bill Smith" are extensional properties. These extensional

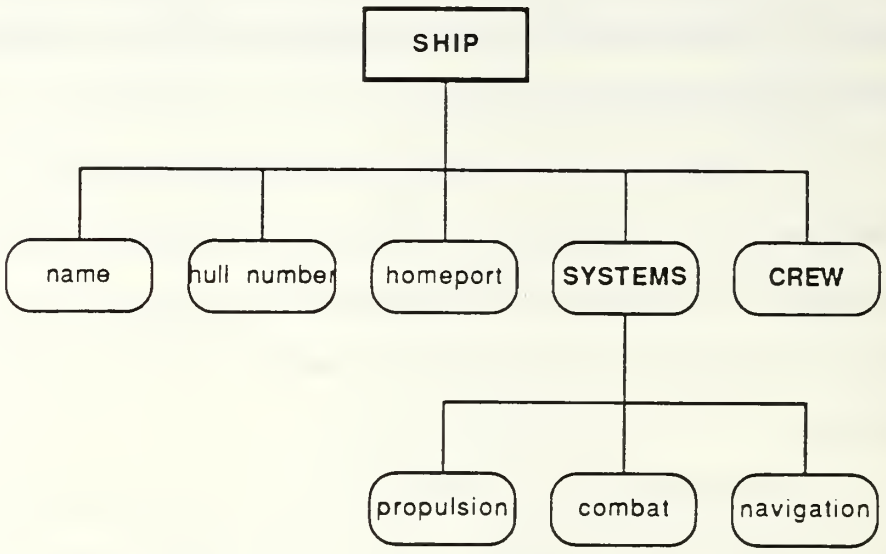


Figure 11. Aggregation

properties represent the actual values for the intentional properties depicted in Figure 11 above.

(3) Association. The *association* abstraction relates similar objects as a higher level set object [Ref. 16]. In this abstraction, the attributes of the set object are emphasized while details of the set members are ignored. Figure 12 gives an example of the association abstraction: here the set object **SHIP** is composed of an association of **crew members,** each of whom has a **name, rank/rate** and **division.**

(4) Version Generalization. *Version generalization* is an abstraction concept in which an object *version* is related to a higher level object, known as an object

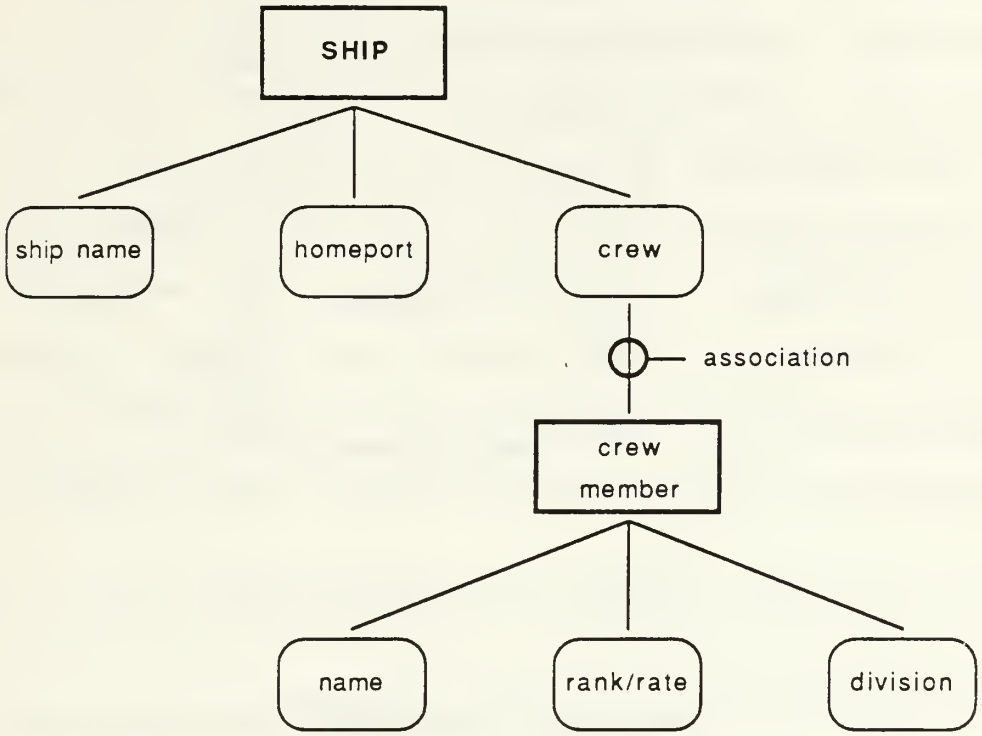


Figure 12. Association

type. Object versions are defined as objects which share the same interface but have different implementations [Ref. 17]. An object type is an abstraction of the common properties of its versions [Ref. 6]. Figure 13 helps to explain these definitions. **Submarine** and **surface ship** are object types which are related, as shown, to object versions **637 class, 688 class, destroyer, aircraft carrier** and **cruiser.** In this example, there is no such thing as a minesweeper.

In this abstraction, versions can have two types of attributes; those which are held in common with the set object and those which are unique for that

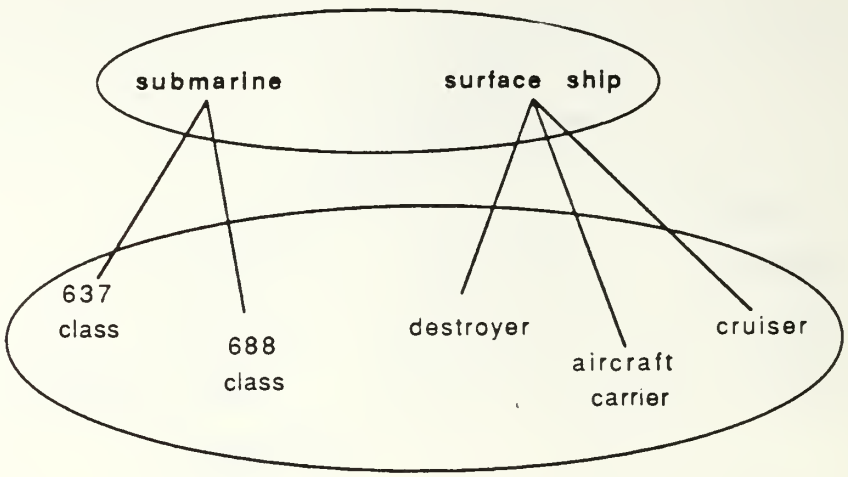


Figure 13. Version Generalization

version. All **submarines,** including **637 class** and **688 class** have the attribute "nuclear"; however, not all **submarines** have the attribute "under-ice capable", as all **637 class** do. Attributes common to both the object type and the version type define the interface characteristics of the object type. Attributes unique to one version distinguish one version from another.

In version generalization, inheritance of attributes is possible, similar to the generalization abstraction. The two abstractions differ, however, in that version generalization specifies the relationship between an object type and its version types, while in generalization, the abstraction is used to specify the relationship between its types and subtypes [Ref. 6].

(5) Instantiation. *Instantiation* produces copies of an object, both object versions and object types [Ref. 17]. Both object versions and object types can be

instantiated. Instantiating a version provides a local working copy of a previous design, with both the implementation and the interface copied. The working copy can be specified to any level of detail [Ref. 6]. Types can be instantiated to produce a working copy where no previous design existed. In this case, no implementation is specified; only the interface is copied. Figure 14 shows an example of instantiation. In this example, the object **CDR Reichel's** ship is an instance of type **SHIP. CDR Reichel's** ship could now be used to produce a working copy of type **SHIP** which could then be used as a starting point for a new design. Since **CDR Reichel's** ship is an instantiation of a type, no implementation details have been provided and this particular instance will be developed from scratch. In the case where **CDR Reichel's** ship is instantiated from the version **San Diego** -based ship, of type **SHIP,** implementation details have been provided

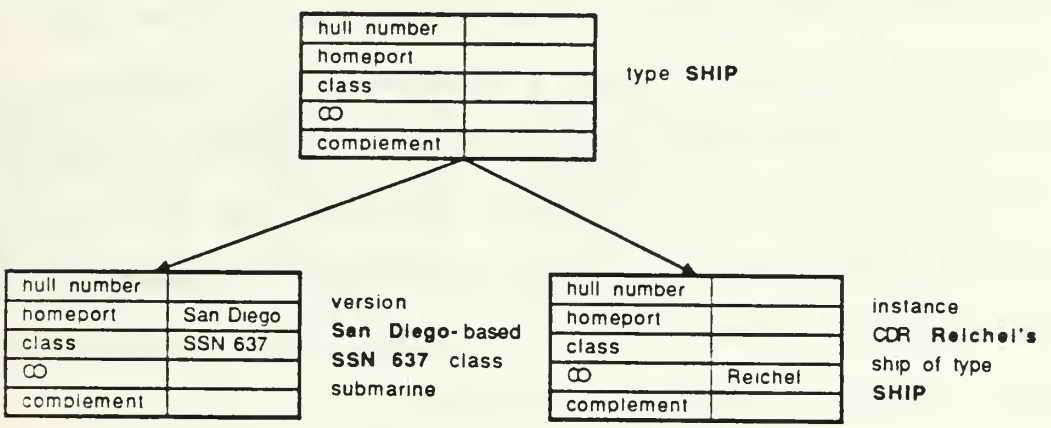


Figure 14. Instantiation

as well as interface details. Here the design of **CDR Reichel's** ship would begin where the implementation details of **San Diego** -based ship left off. This implies that **CDR Reichel's** ship and **San Diego** -based ship have similar implementations.

Instantiation provides for attribute inheritance. An instance of an object version will inherit the attributes of both the object type and the object version. For example, if we depicted an instance of object version **San Diego** -based ship and called it **CDR Wynne's,** this new instance would inherit all the attributes of both **San Diego** -based ship and **SHIP.**

(6) Version Hierarchy. *Version hierarchy* is defined as the hierarchy formed from the set of versions for a particular type or subtype [Ref. 18]. As we proceed from one level to the next lower level in this hierarchy, additional implementation details would be provided. The difference between ordinary generalization and version generalization is that different versions of an object will have the same set of properties with potentially differing values, whereas different types will have different sets of properties [Ref. 6]. Figure 15 shows a representation of a submarine as a version hierarchy. Here, **NUCLEAR** is a subtype of type **SUBMARINE** and "East Coast" and "West Coast" are subtypes of **NUCLEAR.** Each subtype may have its own version hierarchy; in the example, "New London-based" and "Charleston-based" are two mutually exclusive versions of subtype "East Coast." Each block in the example is capable of acting as the initial point for a new design.

(7) Instance Hierarchy. *Instance hierarchy is a hierarchy of different instantiations* of the same types/sutypes or versions [Ref. 18]. Figure 16 depicts an instance hierarchy for the construction of an submarine. The submarine is being built from scratch; that is, no previous drawings are to be used. The starting point for the design is an instantiation of the subtype **NUCLEAR.** As the design progresses, the designer may not be sure whether he wants a state of the art combat system or he wants

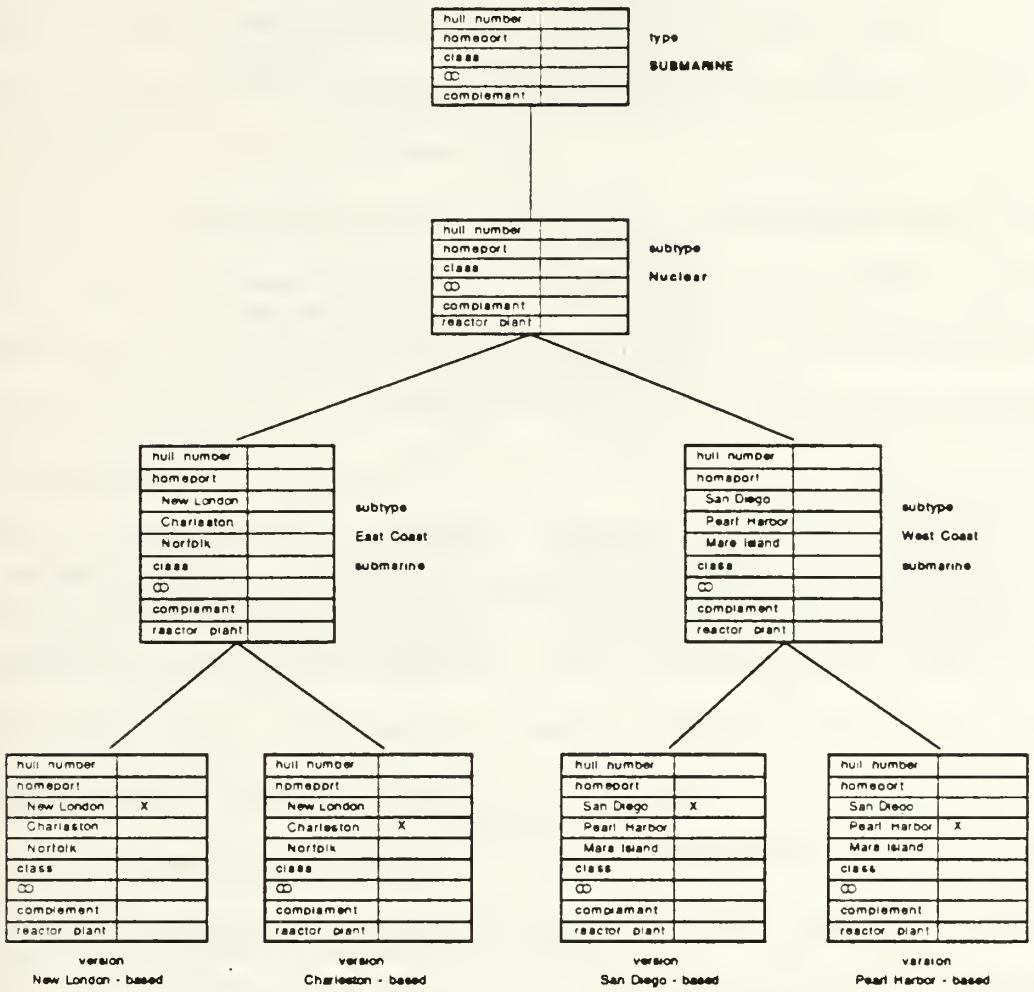


Figure 15. Version Hierarchy

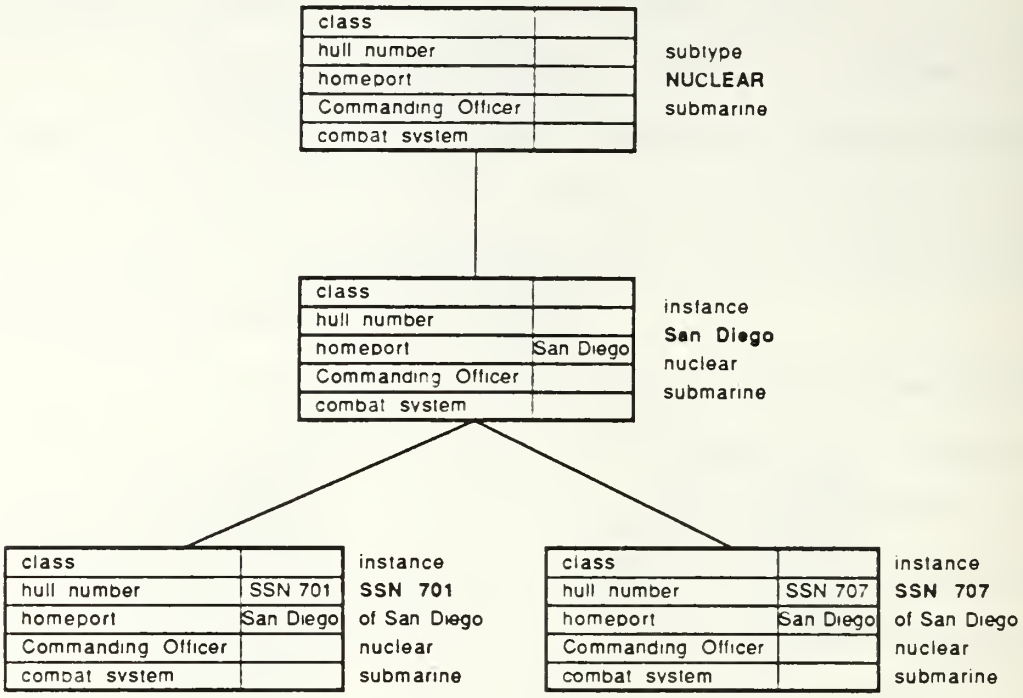


Figure 16. Instance Hierarchy

to use a proven model. The design process can continue on the path he prefers now and, since the instance hierarchy is saved, if he later changes his mind, the hierarchy allows him to go back to the original design and make the necessary modifications.

c. Current Semantic Models

The abstraction concepts we have described in the previous section are combined and redefined in several ways to develop semantic data models. These models use primitives such as objects, entities, and events along with methods for combining these primitives and specifying attributes. Some models, known as extended data

models, integrate programming language constructs into database concepts. They also use advanced concepts such as abstract data types and strong typing [Ref. 6]. Some of the more prominent semantic data models in use include the Entity-Relationship (ER) Model, the Functional Model, SAM*, RM/T, SHM+, SDM/Event Model, and TAXIS. Of these, the latter three are extended data models. We will examine the attributes of these models in the following paragraphs.

(1) Entity-Relationship (E-R) Model. The *Entity-Relationship (E-R)* Model is based on tables and graphs, an outgrowth of the process of designing databases [Ref. 7]. This model bears some resemblence to the hierarchical and network data models, in fact, the E-R model uses a network representation to depict entities as nodes and relationships as edges connecting appropriate nodes. In this model, four levels of views are designated which support both logical and physical database design. These four levels define conceptual objects and their relationships, as well as schema for organizing and storing these relationships. One strongpoint of this model is that it supports many-to-many relationships. To date, its primary use has been in systems analysis and design of databases [Ref. 7].

(2) Functional Model. *Functional database models* seek to represent entities and the relationships between them in terms of the mathematical notion of a function, as a mapping of one object onto the domain of another object. This model treats data definition and data manipulation as integrated. There is no concept of a record or tuple in this model. Instead, the model treats the data together with the operations on the data, similar to an abstract data type.

Functional database models use concepts which are intuitively easy to grasp. These concepts have evolved from mathematics and programming languages and give functional models their power.

(3) SAM*. *SAM** is an improved version of an earlier data model, known as SAM [Ref. 19]. SAM* is a powerful model, capable of supporting temporal, positional and procedural relationships, as well as hierarchies of objects, multiple versions of objects, recursive definitions of objects and complex data types. The model makes use of two types of concepts, *atomic* and *non-atomic*. Atomic concepts are those which cannot be further broken down, are well-defined and understood, and do not need to be defined in terms of other concepts. Conversely, non-atomic concepts are defined in terms of either atomic concepts or other non-atomic concepts. [Ref. 6]

Groupings of atomic and non-atomic concepts are called *associations* and are used to describe higher-level non-atomic concepts. SAM* supports membership, aggregation and generalization associations, analogous to the classification, aggregation and generalization abstraction concepts [Ref. 6].

(4) RM/T. RM/T, or *extended relational model,* is an improvement on the relational data model, providing for null values, support for the aggregation and generalization concepts, and a more diverse group of objects [Ref. 20]. This model represents types as relations with a special internal identifier to depict each instance of the type. Attributes are similarly represented, with the special internal identifier containing property values.

(5) Extended Semantic Hierarchy Model. The *extended semantic hierarchy model,* or SHM+, provides a storage model which attempts to integrate the fundamental concepts of (semantic) data models with the concepts currently polular in the design of programming languages [Ref. 10]. This model is based on an object oriented, rather than a record oriented approach such as that used in the relational data model. SHM+ uses the modelling concepts of classification, aggregation, generalization, and association. It is referred to as an *extension* of the relational model because it provides additional domains and data types, permitting modelling of complex models.

Additionally, SHM+ has the capability to define type hierarchies and provide for inheritance. The hierarchies are composed of instances of subtypes of the parent type. These hierarchies may, in turn, be hierarchies themselves [Ref. 6].

(6) TAXIS. The *TAXIS* data model is the culmination of an effort to integrate tools for the design of information systems [Ref. 21]. The model is object-oriented, employing objects to represent real-world (application) entity, and incorporates the aggregation, classification and generalization abstraction concepts [Ref. 6]. Complex entities can be modeled in TAXIS using a grouping known as a *transaction.* Transactions can be arranged hierarchically to model higher level procedures. TAXIS has a compiling feature, as well, which allows the model to operate much like a traditional relational database management system.

(7) Object-Oriented Models. *Object-oriented* models are distinguished from classical approaches by their ability to handle data of an arbitrary type. Whereas classical approaches handle only data formatted as a record, the object-oriented systems define types similarly to abstract data types in which data and the operations on that data are packaged together. The object-oriented models are based on the classification abstraction concept where objects are grouped into classes based on certain properties. The classes can then be organized into hierarchies which determine inheritance characteristics. [Ref. 6]

III. THE MANUFACTURING DATA MODEL

A. INTRODUCTION

As discussed in Chapter 2, there are three approaches to Computer Integrated Manufacturing (CIM): the high level approach which utilizes expert systems and data translators to support the manufacturing functions, the centralized database approach which uses a central or distributed database to serve the data needs of the manufacturing functions, and the low level approach, which passes data between the manufacturing functions through the use of a data model that fits the needs of each of the manufacturing functions. We listed the advantages and disadvantages of each approach and indicated a preference for the low level approach, in spite of the cost and need for long term planning associated with this method. In this chapter, we will review a model which exhibits many desirable characteristics in supporting CIM, and potentially Flexible Manufacturing Systems (FMS).

C. Thomas Wu and Dana E. Madison of the Computer Science Department at the Naval Postgraduate School have developed a data model in support of their work in databases and Computer Integrated Manufacturing. Since we are describing their model in this chapter, in preparation for applying it to FMS in Chapter 5, this chapter is paraphrased directly from [Refs. 1, 2, 6]. We introduce no original material in this chapter.

B. DATA MODEL DESCRIPTION

Wu and Madison's data model, hereafter called the Manufacturing Data Model, takes a unique approach to modeling the data needs of CIM. Rather than describing the data needs of each process within CIM, i.e., the data needs of Design, the data needs of

Planning, etc., they look at the data needs of the overall manufacturing system. They refer to their approach as data-oriented rather than process-oriented and cite its advantage of integrating all the manufacturing functions within a system, rather than merely automating the interfaces between functions [Ref. 6]. We will begin our review of the Manufacturing Data Model by discussing the abstraction concepts it supports.

The Manufacturing Data Model includes the molecular aggregation, generalization, version hierarchy, instantiation and instance hierarchy abstraction concepts [Ref. 1]. The top level *conceptual schema* is used to depict several of the modeling concepts used in the Manufacturing Data Model. Figure 17 shows a conceptual schema for a **SHIP** and portrays allowable type/subtype aggregations, component relationships, and acceptable combinations of primitive objects which can be manipulated to produce higher-level objects. The conceptual schema defines the primitive objects under consideration. Primitive objects are the low level entities which are manipulated by the data model to support an application such as design, planning, or manufacturing. Primitives can, in fact, be composites of other primitives and can be defined to various levels of abstraction [Ref. 6].

In the Manufacturing Data Model, each type and subtype depicted in the conceptual schema have an associated *prototype,* and within each prototype exist *slots.* These slots contain specific attribute values, or default values, and hold inheritance data. If an instance is desired, an extension of the prototype is created with unique attribute values inserted into the slots.

By assigning specific attribute values to the schema in Figure 17, we could create an instance of a ship, i.e., a specific class or hull number, depending on how detailed the schema was and how specific the attribute values are.

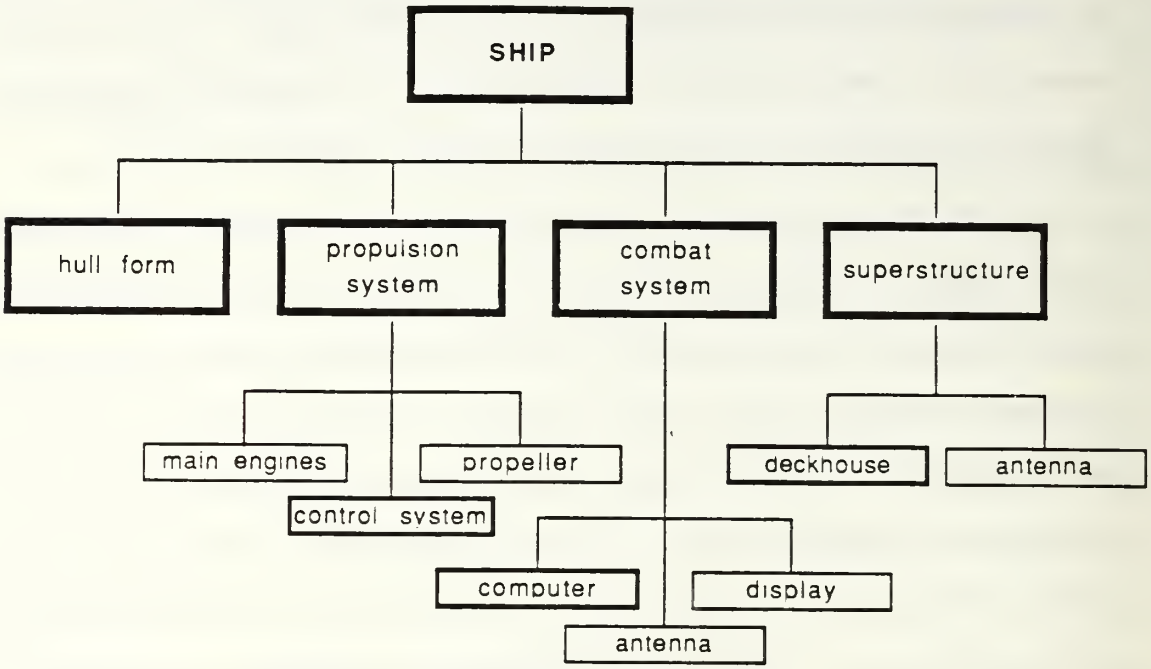


Figure 17. Conceptual Schema of Type **SHIP**

Our generic **SHIP** is an aggregation of a **hull form,** a **propulsion system,** a **superstructure** and a **combat system**. **Propulsion system, superstructure** and **combat system** are also aggregations of objects, in some cases, sharing objects. Both **superstructure** and **combat system** share an object called **antenna.**

Following the notation introduced by Madison [Ref. 6], bold rectangles in the conceptual schema depict types with named subtypes. For example, **propulsion system** can have subtypes such as nuclear, gas turbine or conventional steam. These subtypes are capable of being instantiated to produce a unique ship.

The conceptual schema represents an important segment of the data modeling process. While not part of the data model, it is a medium through which the model may

capture data for an application. The model, in concert with the conceptual schema, represent the range of alternatives available in modeling a particular application. [Ref. 6]

1. Molecular Aggregation

Wu and Madison use the aggregation abstraction described in Chapter 2 to support several manufacturing functions, including product design, where it can be used to model assemblies and subassemblies, and planning, where it can be used to develop process plans.

The Manufacturing Data Model uses aggregation to combine intensions and extensions of objects of potentially dissimilar types into a higher level object, which will turn out to be an intension or extension of a type [Ref. 6]. Figure 18 shows an example of an aggregation of intensions using the Manufacturing Data Model.

2. Generalization

Wu and Madison use the generalization abstraction concept to indicate the relationship between types and subtypes. They treat types as generalizations of named subtypes, which are then treated as primitives which are capable of being made into versions or instances. Figure 19 shows a type **propulsion plant,** created from subtypes **nuclear, gas turbine** or **conventional steam.** In their model, the idea of subtype is important because they allow different subtypes to have different sets of attributes. [Ref. 6] In Figure 19, **nuclear** has an attribute "main coolant pump", while neither **gas turbine** nor **conventional steam** do.

An important aspect of generalization as applied in the Wu-Madison model is the inheritance of attributes between types and subtypes. In Figure 19, **propulsion plant** has been created with attributes **A/C plant** and **distilling plant.** Each subtype, **nuclear, gas turbine,** and **conventional steam** have these same attributes, plus additional attributes which may be uniquely defined for that subtype. The subtypes **nuclear, gas**

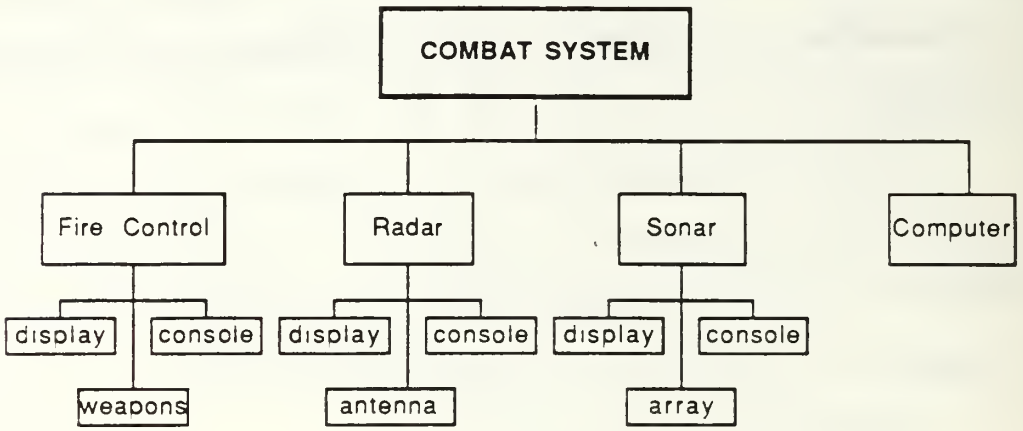


Figure 18. Example of Aggregation

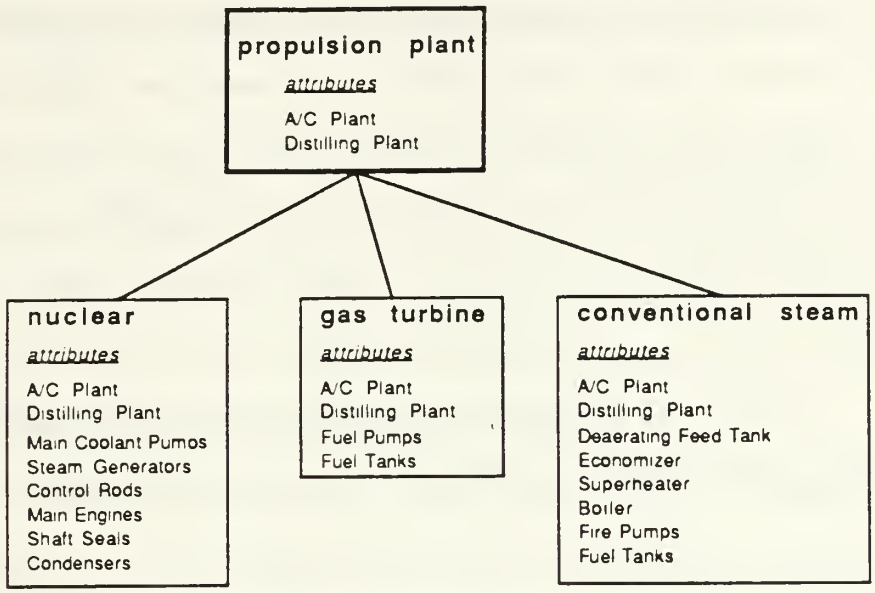


Figure 19. Example of Generalization

turbine and **conventional steam** have their own attributes which define them as unique, as well as any attributes which they inherit from their generalized type. [Ref. 6]

3. Version Hierarchy

Wu and Madison define a version of a type as a molecular object with two objects, an interface and an implementation. The interface for a version is specified by listing properties and attributes which describe the version. In their model, the implementation for a version is specified by providing values for the attributes listed for the interface. The Manufacturing Data Model has all of its interface attributes specified but may have its implementation details in some stage of completion. By defining their model in this manner, a version may be plugged, unplugged or partially plugged [Ref. 17]. In Figure 20 we show an object of type **SHIP** with its interface

defined, as represented by the upper block in the figure with attributes **speed, length, beam, range** and **Commanding Officer** displayed. The implementation details for this object are not specified, as depicted by the missing values for the attributes listed. Object **SSN-XX** has the same interface details as its object type, **SHIP**, as well as some implementation details, indicated by the values filled in for the **speed, length** and **beam** attributes. In this example, the interface (function) of the **SHIP** is specified, while the implementation details (e.g. what is the speed of the ship?) are not fully specified. [Ref. 6]

Versions may have two types of attributes. One type of attribute is that which is inherited from the object type, while the other type are those which have unique values

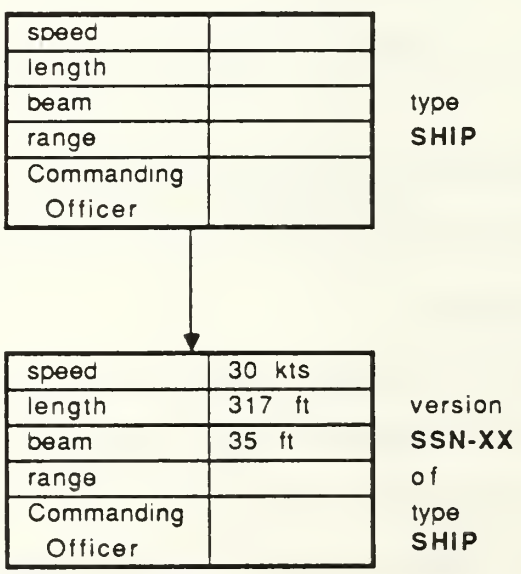


Figure 20. Example of a Version of a Type

for each version. Those attributes inherited from the object type designate the interface characteristics, or function, of the version. The attributes which are version specific are those which differentiate one version of a particular type from another version of the same type. [Ref. 6]

An important note is the difference between a version and an instance of a type or subtype. A version is created at some midway point in the modeling of an application, allowing further work to begin at that point. Implementation details are partially specified. On the other hand, a type or subtype indicates a starting point in modeling an application, with no implementation details provided. [Ref. 6]

The version hierarchy is formed when various values are assigned to the attributes, resulting in a set of possible starting points for future work. This important difference between the Manufacturing Data Model and other data models minimizes the amount of redundant work necessary in all manufacturing functions. Rather than starting each design, or process plan or schedule from scratch, the Manufacturing Data Model concept of version hierarchy allows the designer or engineer to pick up work at some midpoint previously defined.

The definition of version is what gives the Wu-Madison data model this unique capability to pick up an earlier design or plan and continue development from that point. While the original definition of version [Ref. 17] allowed versions to be objects with the same interface but different implementations, Wu and Madison have defined version in a more general manner. In their definition, implementation can be specified to any desired level of detail. For instance, implementation may be plugged, or completely specified; partially plugged, or partially specified; or unplugged, the case where no implementation details are specified. They feel that by generalizing the definition they gain considerable flexibility, allowing them to better model the manufacturing environment. [Ref. 6]

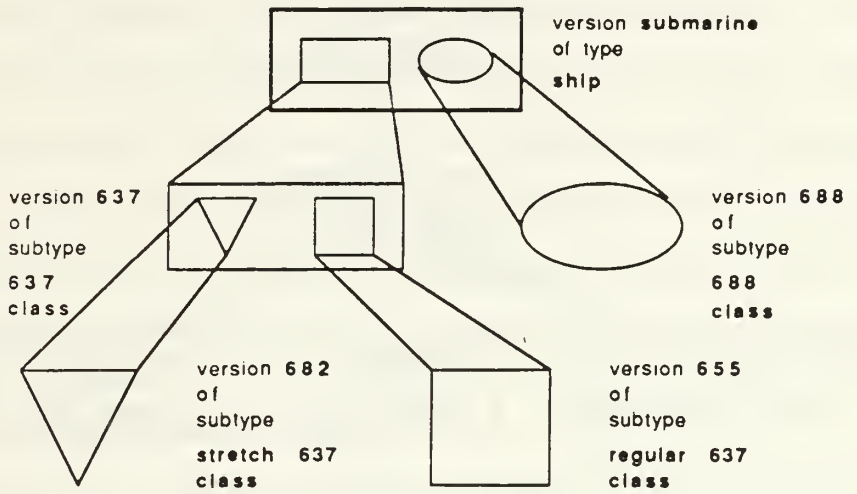
The Manufacturing Data Model version hierarchy is also unique in that it is formed from the specialization of versions to form lower level versions. The existing concept of version hierarchy [Ref. 17] is exactly the opposite; the hierarchy forms from the aggregation of versions to create higher level versions. These two concepts are depicted in Figure 21.

The Manufacturing Data Model has one additional characteristic which gives it the capacity to model a particular application environment. Their model consists of versions which are all of the same type, related as depicted in the version hierarchy. Versions are related to their type by version generalization where the the highest version in the hierarchy is directly related to its type and the lower versions are related indirectly. They feel this provides their model with additional flexibility in relating and representing versions. [Ref. 6]

4. Instantiation

The Manufacturing Data Model makes use of the instantiation abstraction concept to create versions and instances of objects. By instantiating types and versions, it is possible to produce either an instance of an object or a new version.

As with the generalization abstraction concept, instantiation provides a similar means of inheritance. Instantiation inheritance duplicates all of the attributes and attribute values of the object being instantiated. New attributes may not be identified but attribute values may be further specified. Figure 22 shows this inheritance. In this example, the attributes of **SSN 637 class, SSN 666** and **SSN 667** are the same as the attributes of type **SUBMARINE.** The differences between these instantiated objects, either versions or instances, are differences in attribute values. **SSN 637 class** will have only a few attribute values specified, whereas **SSN 666** and **SSN 667** will have all their attributes specified, many with different values. Wu and Madison point out that



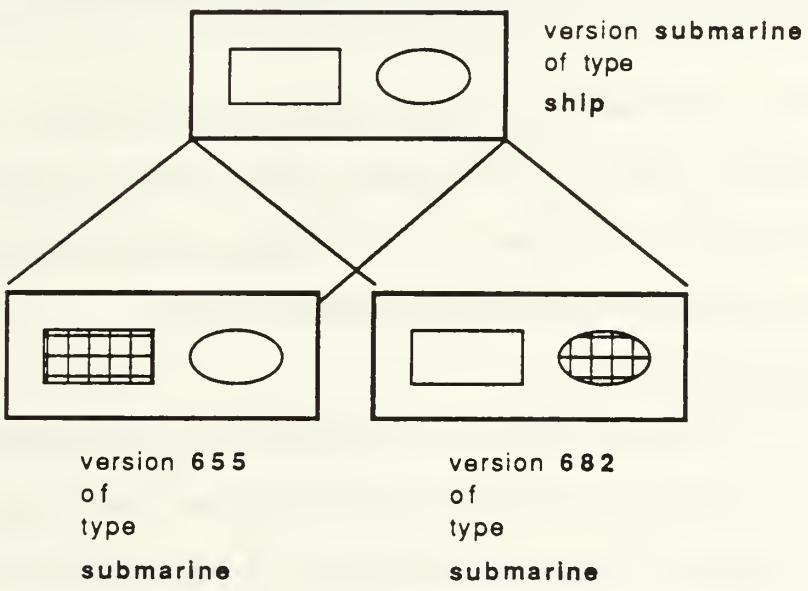


Figure 21. Comparison of Version Hierarchies

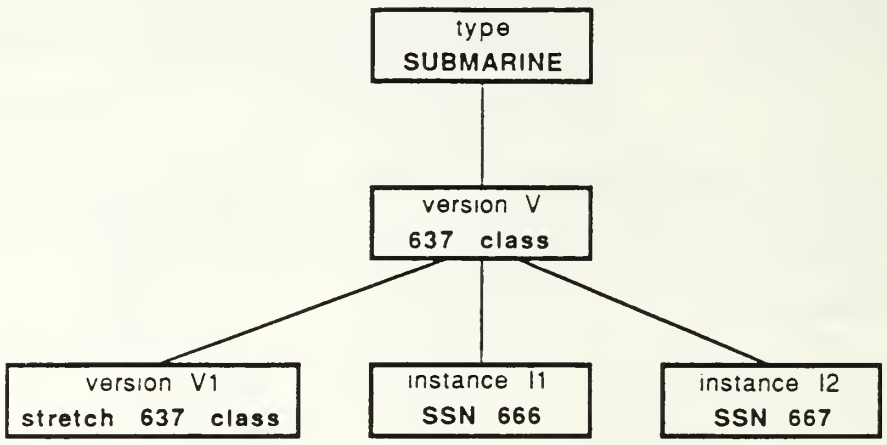


Figure 22. Example of Instantiation Inheritance

instances, such as **SSN 666** and **SSN 667**, represent real world objects while versions, such as **stretch 637 class** serve as templates which define real world objects to a particular level of detail. In Figure 22, we could add an instance I3 below version V1 to represent an real world instantiation of this template. [Ref. 6]

5. Instance Hierarchy

Wu and Madison introduce a new abstraction concept in their Manufacturing Data Model, the instance hierarchy, to complement the other concepts which they employ. The instance hierarchy allows the user to archive previous instances of a particular object. They stress that the instance hierarchy is a temporary entity within the system, pending a decision by the user as to which particular instance (design, process plan, schedule, etc.) he wishes to implement. In Figure 23 we have modified an example

from [Ref. 6] to depict the operation of the model as an engineer designs a submarine. As shown, once the engineer chooses the alternative in the hierarchy to become the final ship design, the hierarchy is collapsed, leaving only the selected design. In our example, he has chosen a **stretch 637 class** submarine as the design he intends to work with. He could also have chosen create a new design from the selected version. In Figure 23, we have shown a situation where the engineer has decided to create a new version of the **stretch 637 class,** the **special DSRV-configured stretch 637 class.** This design is added to the version hierarchy to be used as a starting point for future work. In this case, attribute values must be considered for specification when the version is created.

C. FORMAL MODEL DEFINITION

Wu and Madison use formal mathematical notation in [Ref. 6] to define their data model. We point this fact out to the reader in the event further background in the model is desired. We will not review the mathematical definition here as it is not pertinent to this thesis.

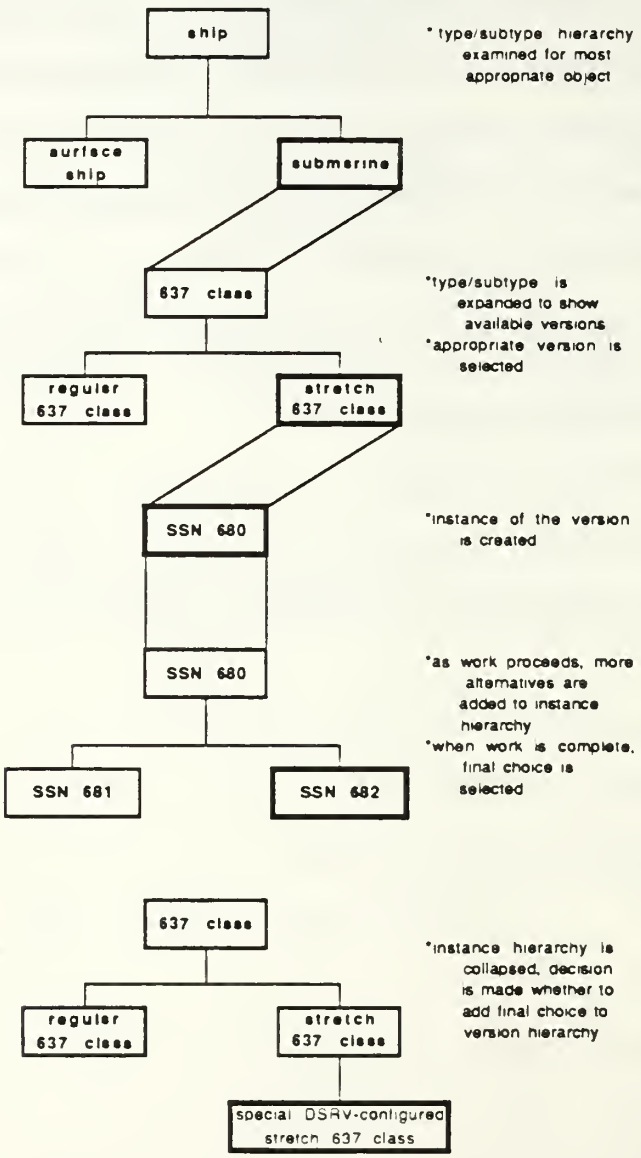


Figure 23. Operation of the Manufacturing Data Model

IV. THE FLEXIBLE MANUFACTURING SYSTEM

A. INTRODUCTION

The Flexible Manufacturing System (FMS), as a manufacturing function within Computer Integrated Manufacturing (CIM), is a set of computer controlled workstations and the transportation components which link them, designed to efficiently produce products at low or medium volumes [Ref. 6]. As a production method, the FMS provides the computer integrated factory opportunities to:

* increase production,
* decrease costs,
* manufacture parts in random order, rather than based on material available,
* decrease inventory and work-in-progress levels,
* provide for inspection of all work,
* decrease the need for repetitive or dangerous work by humans,
* increase the need for intelligent work by humans,
* provide a reprogrammable, and in some cases, unmanned, manufacturing facility for a wide range of items [Ref. 22].

With these positive points, one might assume that FMS technology is the solution to the problems of CIM, in particular, the lack of integration between manufacturing functions. In fact, this is not necessarily true, and in most cases, FMS is still pursuing the goals of CIM. In this chapter, we will review FMS system architecture as it exists today, pointing out a need for integration which closely parallels the needs of CIM presented in Chapter 2.

B. FMS HISTORY

FMS evolved from the Numerical Control (NC) machines of the 1950's and 60's. These machines were the first programmable computer-controlled "robots" in the factory. They operated on a simple scheme involving some type of communication system, often a punchcard, providing input in the form of coded numbers to the machine. The numbers represent the various functions which the machine is capable of performing. For example, the sequence "5 4 6 2" might represent pick up tool "5", move left "4" inches, cut to a depth of ".06" inches, then perform the sequence again.

The next step in the development of FMS was the emergence of the Direct Numerical Control (DNC) machine in the mid 1960's. The DNC machines coupled more powerful computer control systems with tool handling and material handling systems.

The first direct ancestor of today's FMS came in 1975 when a Numerical Control machine was mated with an Automatic Tool Changer (ATC) system, a pallet pool, and an Automatic Pallet Changing (APC) system [Ref. 22]. From this point on, the technology evolved rapidly. Today, FMS is not only used in machining, but in welding, cleaning, painting, inspection and packaging. The technology is even spreading beyond the spectrum of manufacturing into non-traditional CIM environments. The Japanese envision an automated textile factory, integrating garment design and manufacture around an FMS [Ref. 22]. In this facility, design data would be used directly by cutting and sewing machines to produce garments with a minimum of human intervention.

The importance of modern manufacturing concepts such as FMS becomes apparent when one considers industry and population trends. Manufacturers will replace their current manufacturing methods and machines by the year 2000, not piecemeal, but *in total.* Today's young people prefer working in service industries, rather than manufacturing, reducing manufacturing's employee base. These facts indicate that

productivity and efficiency will become even more important in the future of manufacturing. FMS, with its impressive list of assets cited above, will be in the forefront of this manufacturing revolution. [Ref. 22]

C. FMS SYSTEM ARCHITECTURE

Most FMS have a common architecture, built around a set of basic rules.

1. Ranky's Rules

These architecture rules, identified in [Ref. 22] give top-level guidance in designing a productive and efficient FMS. The rules make use of the concept of a "cell", the smallest, single-function component of the FMS.

a. Cells

The FMS should incorporate automated and programmable machines, or cells, capable of operating unmanned, utilizing ATCs and self-diagnostics while coupled to a central computer which provides machine programming and data. These cells can range in function from machining to inspection to packaging.

b. Transportation System

The cells should be connected by a system which provides material or parts access in a random (Automated Guided Vehicle) rather than serial (conveyer belt) manner.

c. Storage Facility

The system should incorporate a parts, tools and pallet storage facility.

d. Computer Control

A distributed processing system should be implemented to provide computer control of databases as well as links to external entities such as Computer Aided Design (CAD), Computer Aided Manufacturing (CAM) and all related business functions.

e. Reliability and Flexibility

The system should be designed so that, in the event a particular cell breaks down, sufficient redundancy and flexibility is present to allow for continued operation.

2. System Configuration

Using these basic rules, it is possible to produce a typical list of cells which might comprise a FMS.

a. Control System

The control system includes facilities for operator interface and FMS control and programming. This system could include features such as editing, a CRT display, diagnostics and system control software.

b. Functional Cell

The FMS contains one or more functional cells, components which determine the purpose and capabilities of the system. Typical functional cells include machining cells, inspection cells, part washing cells and painting cells.

c. Transportation System

The functional cells would be linked by some type of automatic work changing system, capable of interchanging palletized workpieces. The system would also present the palletized workpieces to an AGV system, the link from the FMS to the outside world.

d. Automatic Tool Changing Cell

This cell interacts with the functional cells, replacing tools as necessary. Typical tools which are changeable include chisels, blades, chucks and bits.

e. Storage Facility

The FMS requires a warehouse to store extra pallets, tools and material stock. In general, this warehouse is an automated activity, interacting with AGVs.

f. An Example of FMS

Figure 24 depicts a generic FMS, comprised of a machining cell, part washing cell, controller and transportation system. Not shown is the external storage facility. We will use this simple FMS in Chapter 5 when we discuss applying the Manufacturing Data Model.

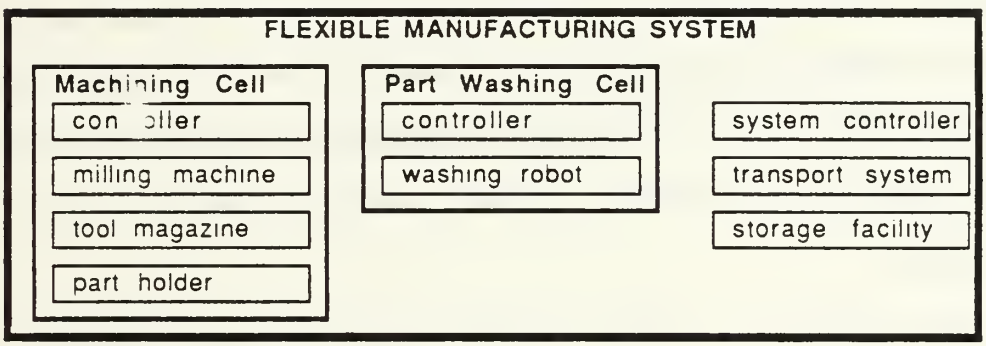


Figure 24. Simple Flexible Manufacturing System

D. SCHEDULING A FMS

Scheduling a FMS is usually approached from a mathematical, or statistical, basis. It should be emphasized here that very effective methods exist to schedule and program the cells of a FMS, however, all current methods treat the FMS as a separate manufacturing function. Integration of the FMS with design, business functions and other manufacturing functions is not considered. Since it is our intention to provide a means to integrate these functions with FMS, we will provide just an overview of the most popular current technique.

Several traditional methods are used to schedule conventional manufacturing shops and several are applied with varying degrees of success to FMS. Of these, the "n" job, "m" machine method extended for FMS is probably the most widely used [Ref. 22]. This method is based on combinatorial mathematics and uses several rules and guidelines to aid the scheduler in writing an efficient machine program. The method is presented in significant detail in, [Ref. 22] therefore, its intricacies and nuances will not be presented here. Instead, since it is the most common method of programming FMS, we will state its shortfalls.

The "n" job, "m" machine scheduling method has three significant problems, discussed below.

* The method is slow in a real world environment. Using combinatorial mathematics, combining five jobs with five machines results in 25 million possible combinations [Ref. 22]. This type of computation is capable of slowing a FMS, normally supported by a mini-computer, to a crawl. Even calculations taking a few minutes may produce a schedule which is already obsolete by the time it is implemented.

* The method does not account for the dynamic nature of a FMS. Events such as material shortages, tool breakages, urgent or modified jobs and computer faults require significant changes in the rules which the method is based upon [Ref. 22]. Again, the method's slow response time will result in the use of out-of-date schedules.

* Modifications are time consuming, expensive and difficult. New rules must be

developed and implemented, then the scheduling method must be run. The result, again, is a schedule which may require further modification to make it current.

These shortcomings, along with the lack of integration common to all forms of CIM systems, are exactly the type of problems the Manufacturing Data Model, presented in Chapter 3, was developed to correct. In the next chapter, we will apply the model to FMS.

V. APPLYING THE MANUFACTURING DATA MODEL

A. INTRODUCTION

In Chapter 3, we introduced the Manufacturing Data Model, a data-oriented approach to *true integration* of the basic manufacturing functions. Chapter 4 reviewed a manufacturing function within Computer Integrated Manufacturing known as Flexible Manufacturing Systems (FMS). In this chapter, we intend to show the power of the Manufacturing Data Model by applying it to a FMS. This thesis will conclude with recommendations for further research in this area.

The Flexible Manufacturing System, as we presented in Chapter 4, is a collection of computer-controlled *cells*, or semi-independent workstations, designed to manufacture an assortment of products at low or medium volumes [Ref. 6]. One characteristic of a FMS is that it is used to produce one product at a time, that is, a particular FMS will only be used to produce a pencil sharpener *or* only be used to produce a can opener. It will not be used to produce both pencil sharpeners and can openers simultaneously. Thus the scheduling problem in the FMS environment reduces to scheduling multiple FMS; for instance, using FMS 1 to produce pencil sharpeners and FMS 2 to produce can openers. This work has been done in [Ref. 2] and will not be discussed in this thesis. Instead, we will focus on using the Manufacturing Data Model to perform the equivalent of process planning on an FMS. We will begin by looking at how the Manufacturing Data Model can be used to develop machine programs for the FMS. Before delving into the FMS, however, we will look at how the Manufacturing Data Model has been applied in previous work. We will begin by looking at the traditional approach to process planning.

B. THE CONVENTIONAL APPROACH TO PROCESS PLANNING

Process planning is the development of a specification defining the operations which must be performed on a part, or several parts, in order to produce a particular finished item. Traditionally, process planning has been performed manually by an industrial engineer, working with design drawings and his knowledge of the plant's equipment. Condiderable effort is made to maximize the efficient use of the plant's equipment, minimizing points where parts back-up, awaiting an available machine ("bottlenecks"), as well as points where machines sit idle. Obviously, this manual approach is very labor-intensive, inexact, and ripe for automation. In fact, automated process planning schemes have been developed but they are not integrated into the complete manufacturing and design functions. They may use specially produced data or translated data from the design process and their output of these automated schemes, rather than being in a form which could be applied directly on the manufacturing floor, must instead be translated again prior to use. While this approach removes much of the human error in process planning and saves considerable time, it should be viewed as a step on the road to true integration. In truly integrated process planning, design data in a specified format is passed directly to a process planning application, processed, and passed on to other manufacturing functions (scheduling, shop floor layout, business applications, etc.) for direct application without translation. The essence of this truly integrated approach is the absence of translators and a minimum of human intervention beyond the boundaries of the particular manufacturing functions. This is the goal which the Manufacturing Data Model attempts to achieve.

C. THE MANUFACTURING DATA MODEL APPROACH

Wu and Madison, in their work in process planning [Ref. 6], approach the subject from a data-oriented, rather a than process-oriented, approach. They view process

planning as divided into four phases. Their first phase involves a gross decision regarding the level of machining or assembly required to produce a given piece. In an example, they consider three possibilities: 1) a piece could be machined from a casting, 2) a piece could be machined from raw stock or 3) a piece could be assembled from smaller parts or assemblies. The first and second possibilities involve parts manufacturing, the stepwise changes to a part as it progresses from an unmachined state to a finished, machined condition [Ref. 6]. The third possibility, assembly, is the combination of two or more parts or sub-assemblies to produce a finished product.

In the Manufacturing Data Model data oriented approach, the second phase in process planning involves selecting and sequencing the appropriate tools and procedures as required by the decision made in the first phase. This phase determines the sequence of steps the part will follow within the shop, from the point where it is delivered to the machine, through the various machining and assembly processes, to the point where it leaves.

The third phase in the process planning example presented in [Ref. 6] selects the appropriate machine tool for each operation chosen in the second phase. For example, if a part is to be machined and then bored, this phase would involve selecting the proper machining equipment (milling machine, grinding machine, etc.) and boring equipment (drill press, clamping pallet, etc.). This phase does not consider the actual availability of machines on the floor, but instead describes process details such as cut and feed rates, cut sequences, cleaning techniques and computation of individual and overall process times.

The fourth phase selects individual tool types for the equipment selected in phase three. If a drill press was selected in phase three, the industrial engineer would select a 3/4 inch bit in this phase. Availability of tools is not a consideration.

Once these four phases are addressed, the industrial engineer portrays alternative process plans for a particular product or set of related products as an acyclic directed graph, depicting all possible choices from each manufacturing activity. Figure 25 shows an example of one of two alternative process plans for the manufacture of a can opener.

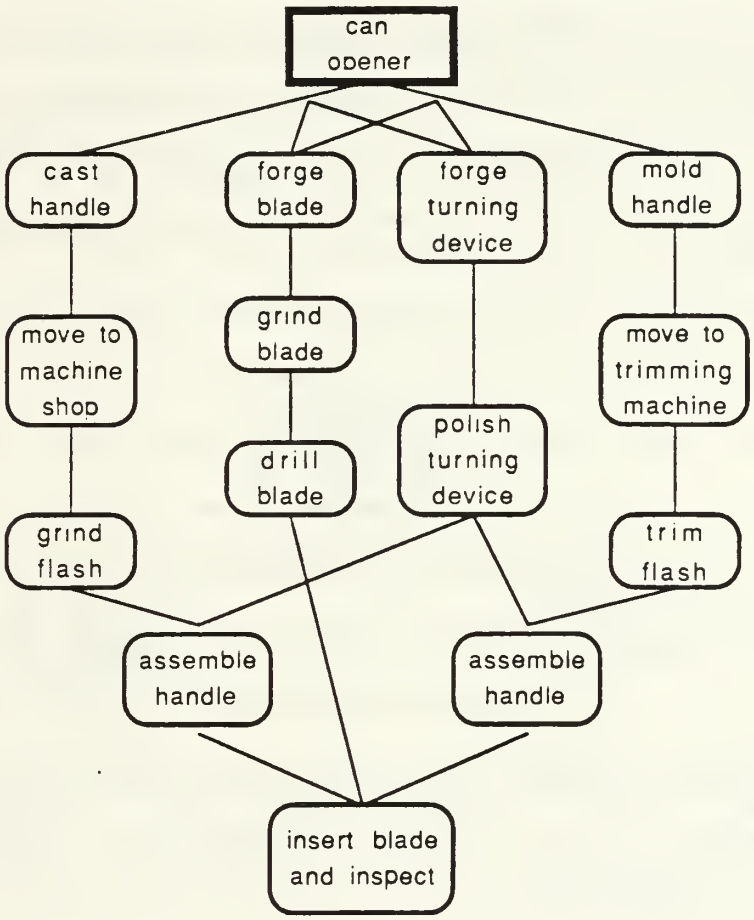


Figure 25. Graph of Alternative Process Plans

Using this example, we will step through the activities Wu and Madison use to semantically describe the process planning evolution.

The industrial engineer can begin development of a process plan either from scratch or by modifying a previously determined process plan. A conceptual schema becomes the basis for an original process plan. This is the same conceptual schema which the Manufacturing Data Model utilizes in product design. The initial step is to create an instance of the type represented by the conceptual schema. An example of such a schema is shown in Figure 26. Once this instance of type **Can Opener** is complete, the development of the process plan shifts to a bottom-up approach. The engineer chooses a component from the lowest level of the conceptual schema and defines the process plans

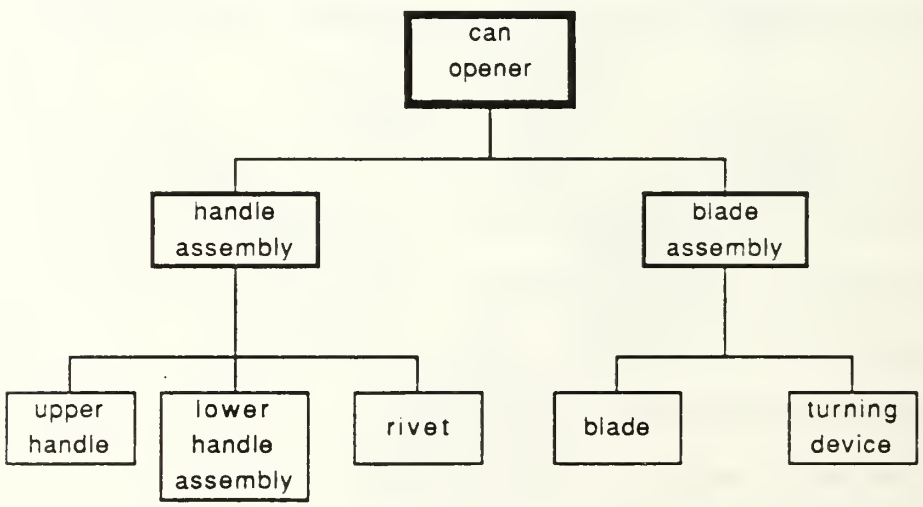


Figure 26. Conceptual Schema for Product Type **Can Opener**

for these primitive components, in our case, the blade, the turning device and the handle. After the lowest level process plans are developed, the plans for the next higher level, the blade and handle assemblies, are defined. The levels are related by the aggregation abstraction concept which implies that a process plan at one level need only consider the process plans at the next lower level, in other words, an assembly procedure. The process plan definition continues upward, level by level, until the top level is included.

The use of aggregation considerably simplifies the process planning procedure in that it parallels the human thought process in product design and manufacture. Humans tend to consider an item as an aggregation of smaller subassemblies.

Definition of primitive level process plans requires the engineer to make a determination as to which information can be specified directly and which must be described later, in the form of parameters. The Manufacturing Data Model can hold these parameter values undefined until the generic process plan is used for production. They cite as an example in their work a machined cut. Machine tool and machine type can be specified during development of the process plan. Dimensions of the cut may remain undefined, or parameterized, until the dimensions of the workpiece are specified and the generic process plan is made workpiece-specific.

If a previously defined process plan is to be modified to produce a new plan, the industrial engineer would begin with the version hierarchy for the type of product concerned. Figure 27 shows a simple version hierarchy for product type **Can Opener.** The engineer would first choose the particular version closest to the process plan he is considering. He would then create an instance of the selected version and modify it, level by level as described above, using the conceptual schema as a guide. The completed process plan would then be added to the version hierarchy, as shown in Figure 28. The Wu-Madison approach to process planning reduces the complexity of this

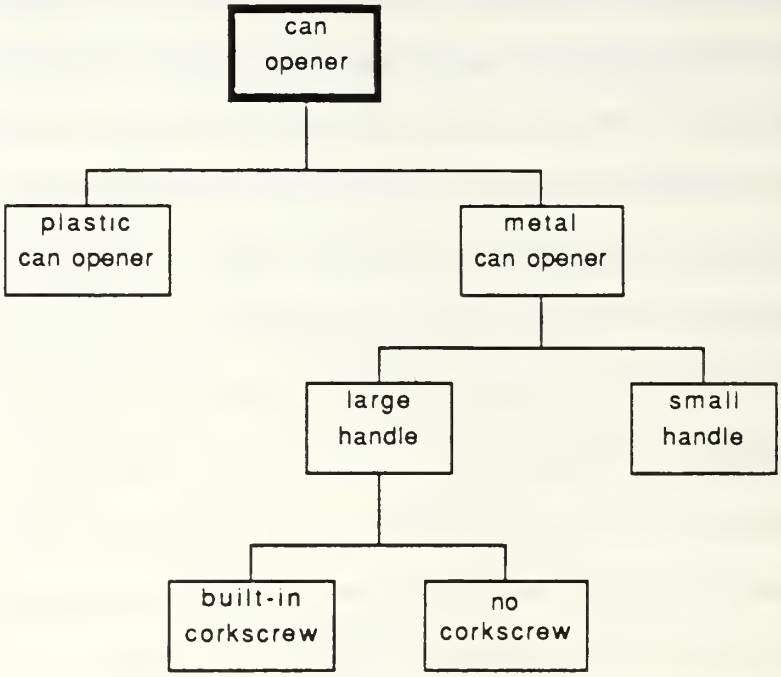


Figure 27. Version Hierarchy for Product Type **Can Opener**

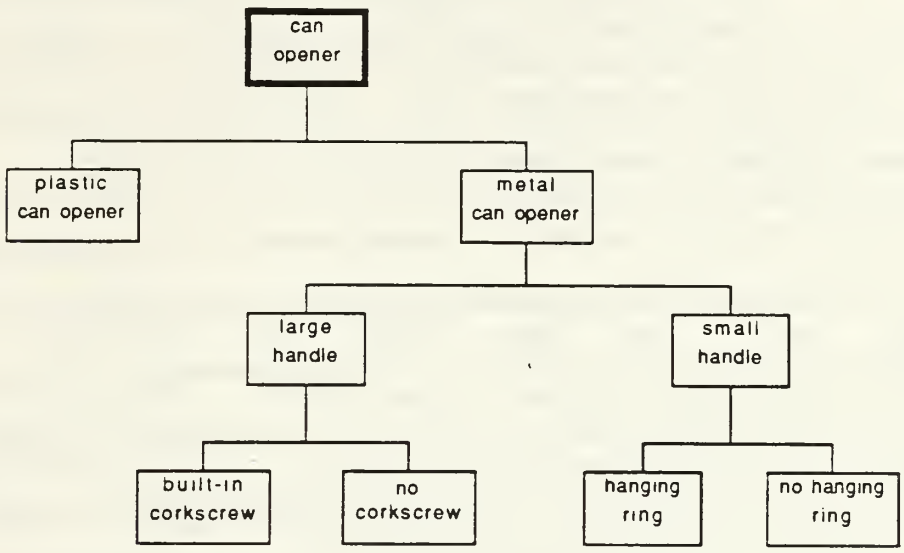


Figure 28. Updated Version Hierarchy for Product Type **Can Opener**

manufacturing function through the use of abstraction concepts and reuse of generic process plans. The model is directly applicable to Flexible Manufacturing Systems.

D. FLEXIBLE MANUFACTURING SYSTEMS

Process planning, scheduling and Flexible Manufacturing Systems are similar in several aspects. In all three cases, the design engineer must integrate the competing demands of two or more process plans with the limited manufacturing resources available. Modeling techniques applicable to one should, with minimal modification, apply to the others. In this section, we will develop a simple FMS, to be used later when we apply the Manufacturing Data Model.

An FMS is composed of several automated, programmable cells, each comprised of related machines. For example, a machining cell may include a controller, a milling machine, a tool magazine and a part holder. A part washing cell might include a controller and a washing robot. These cells are connected by a transportation system, usually some form of robot-operated pallet. A parts and material storage facility, or warehouse, supports the overall system. Figure 29 shows a basic FMS.

Chapter 4 discussed the basis for current scheduling techniques for FMS. These systems are scheduled and programmed using combinatorial mathematical procedures. In general, these techniques are effective in stable environments. However, if change is introduced into the FMS, either in the form of a new product to be manufactured, a disabled cell, or a change in product design, the FMS responds slowly due to the nature of the programming/scheduling process. Some system to use design data directly in the programming/scheduling environment, coupled with a system which could archive

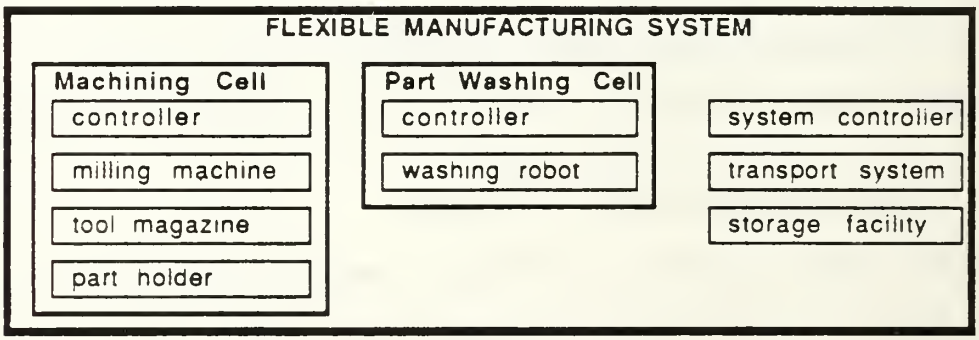


Figure 29. Simple Flexible Manufacturing System

previously used schedules for other products, would significantly improve the efficiency of a FMS. The Manufacturing Data Model offers these capabilities.

E. APPLYING THE MANUFACTURING DATA MODEL TO FMS

The strength of the Manufacturing Data Model is derived from the use of several abstraction concepts to simplify and provide flexibility to the modeling of several major manufucturing functions. In this section, we will address the applicability of the model in support of FMS.

The Manufacturing Data Model makes use of the aggregaton, generalization/specialization, version hierarchy, instantiation and instance hierarchy abstraction concepts to describe the process planning, scheduling and shop floor layout problems. We will use these abstractions to address the FMS programming situation.

Aggregation is the abstraction of a set of objects and their relationships into a higher level object [Ref. 14]. This concept permits the designer to work at the appropriate level of detail, hiding unnecessary details of implementation.

The simple FMS, depicted in Figure 29, consists of an aggregation of a machining cell, a part washing cell, a controller, a storage facility and a transportation station. The machining cell and the part washing cell are aggregations, that is, they are composed of lower level, or molecular, objects. The controller, storage facility and transportation system are molecular objects. In the Manufacturing Data Model, the interface of an aggregation is determined by the interfaces of its molecular components [Ref. 6]. The function, or implementation, of an aggregation is likewise determined by the functioning of its molecular components. In other words, the function of the part washing cell is determined by the function of the controller and the robot.

The generalization/specialization abstraction concept [Ref. 14] is used in the Manufacturing Data Model to relate types and subtypes. Types are generalizations of

subtypes, as described in the can opener example in section C above. Subtypes are an important aspect of the Manufacturing Data Model as different subtypes are allowed to have different sets of attributes. Types and subtypes can function as primitive entities from which versions and instances can be defined.

In Figure 30, we show a generalization of a FMS. In this example, **FMS** is a generalization of **COMAO FMS** (a particular make of FMS) and **other FMS w/i same factory** (representing other FMS within the same factory). Likewise, **COMAO FMS** is a generalization of the specific cells which comprise it. Conversely, **lathe tool** is a specialization of **tool magazine.**

Wu and Madison define a version of a type as a molecular object with interface details specified and implementation details unspecified. By defining a version in this manner, the version can be plugged, unplugged, or partially plugged [Ref. 17]. Figure 31 depicts an object of type **machining cell** with an object version V1 *tapping machine* of type **machining cell.** The object of type **machining cell** has its interface defined, which is represented by the shaded interface box. Implementation, or function, details, however, are not specified as depicted by the unshaded implementation box. Object V1 has identical interface details as its object type **machining cell** and has some implementation, or function, details specified, represented by the partially shaded implementation box.

As we described in Chapter 2, versions are allowed to have two types of attributes. One type is inherited from the object type and passes on the interface characteristics of the object type. Using our example, the *tapping machine* inherits interface characteristics from the object of type **machining cell.** The other type of attributes are those specific to that particular version. These are the attributes which differentiate one version of a specified type from another of the same type.

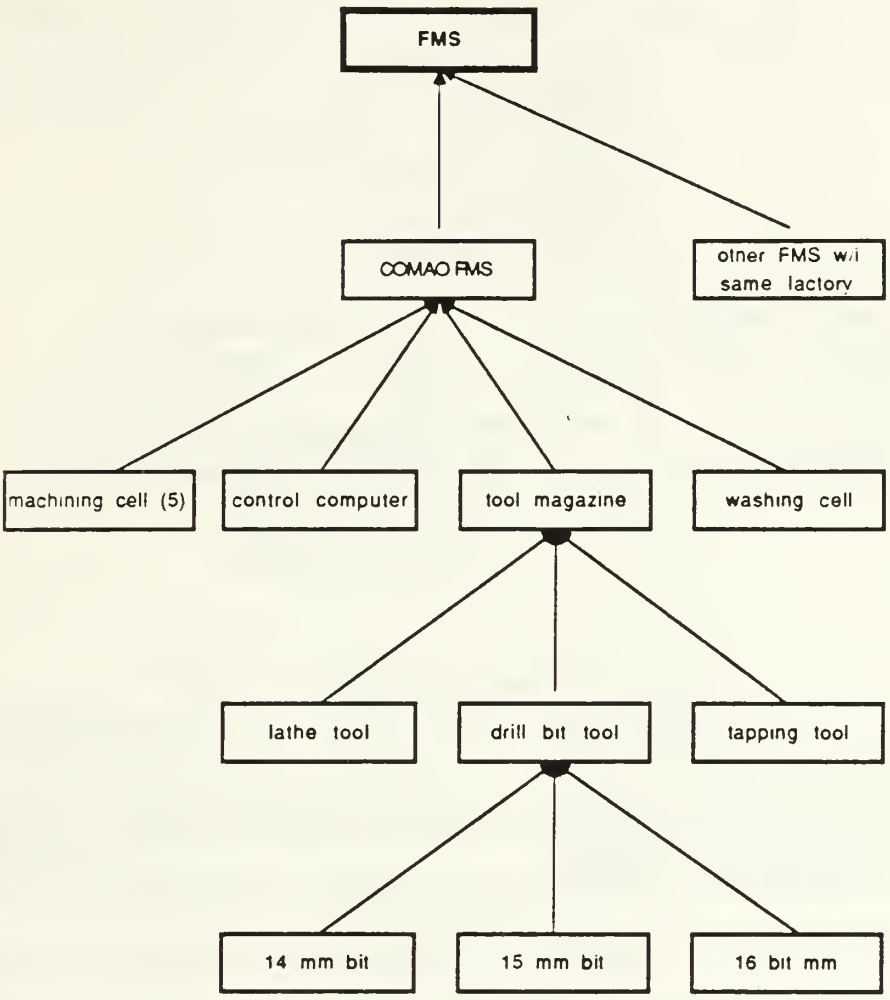


Figure 30. Example of Generalization/Specialization in FMS

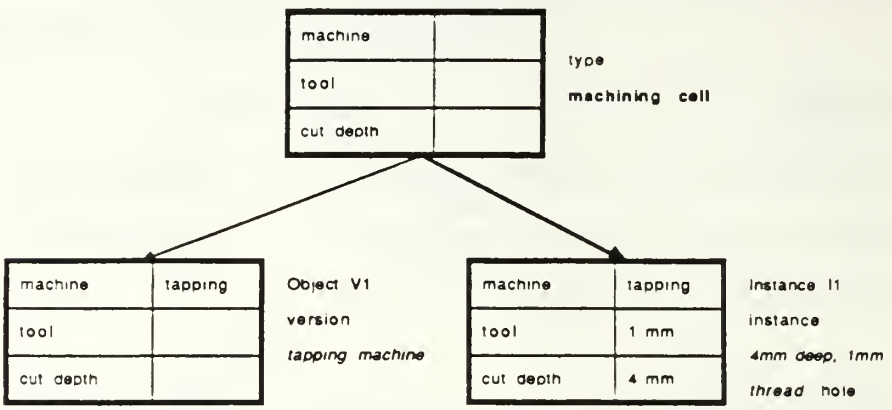


Figure 31. Example of Versions in FMS

Wu and Madison use the idea of parameterized versions to specify implementation details of a particular object. This concept is valuable in defining implementation details for an instance of a type, an object in which implementation details are not specified. In this case, an instance of an object type is produced rather than an instance of one of its versions. This instance of an object type is called a parameterized version. In other words, defining an instance of a specified object type produces a socket which will accept any version of that specified type. Different versions can be plugged into the sockets, creating a unique FMS implementation.

The next abstraction concept used in the Manufacturing Data Model is the creation of an object by instantiation. Object types and object versions are capable of being instantiated [Ref. 17]. Instantiating a type produces a copy of a process plan, or

schedule, or shop floor layout, or in the case of a FMS, a FMS program. In Figure 32, a simple FMS program is depicted as an object type. Also shown is a version of the object type, instantiated to produce a particular product. The object type itself can also be instantiated, as shown.

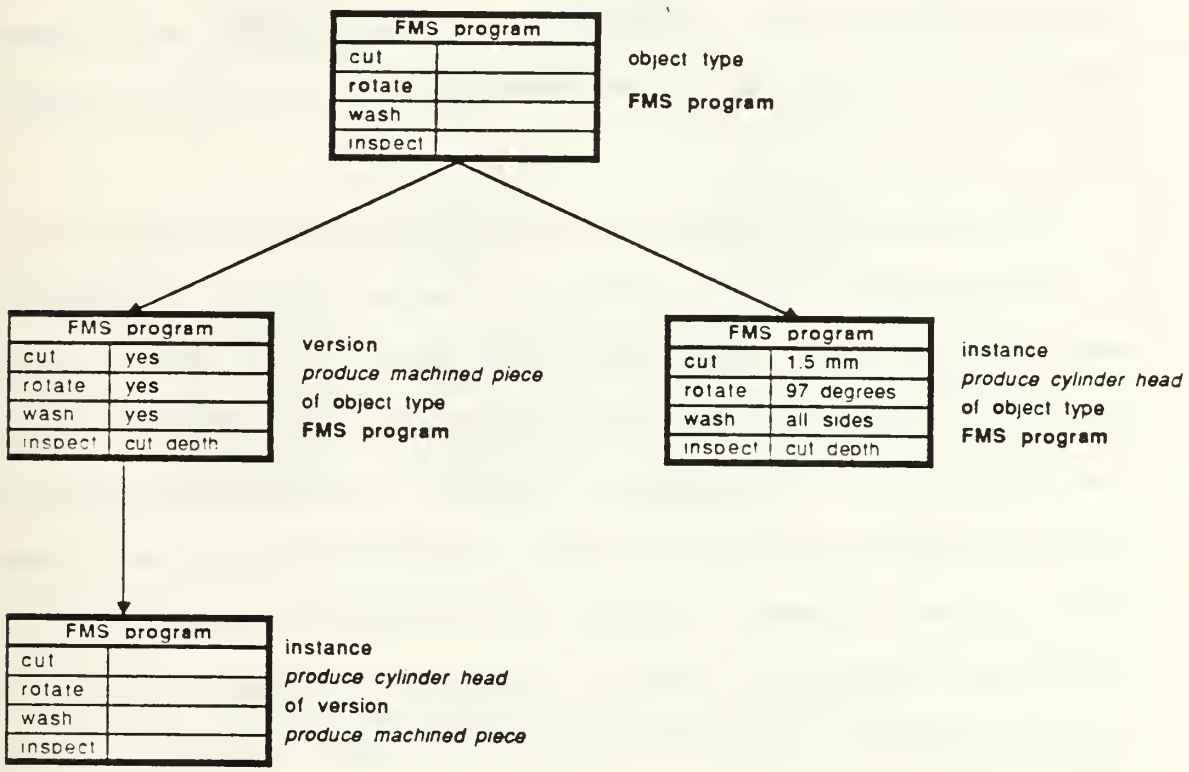


Figure 32. Instantiation

The instantiated object type can be considered a working copy of the object type and will function as the starting point for a new FMS program. Since this working copy is instantiated from its parent type, this implies that implementation (function) details are not specified and must be developed from scratch. In our example, the program on the right in the figure is instantiated from its parent type. The function of this program is awaiting definition.

If a FMS program is instantiated from a version of the object type, such as the program on the left, any implementation details specified in the version would be inherited by the new program. In other words, if the function of the version is *produce cylinder head,* the function of the instantiated version would also be *produce cylinder head.*

The final abstraction concept which the Manufacturing Data Model makes use of is the version hierarchy. The basic definition of a version hierarchy is a hierarchy of versions, with increasing implementation detail specified as you proceed to lower levels [Ref. 6]. As described in Chapter 2, the version hierarchy differs from the type generalization hierarchy is that different versions of an object have the same set of attributes, but not necessarily the same values, while different types may not necessarily have the same set of attributes. In Figure 33, we have depicted a version hierarchy of the FMS program **manufacture crankshaft. Manufacture crankshaft** is a type, while *manufacture 4-cylinder crankshaft* and *manufacture 8-cylinder crankshaft* are subtypes of **Manufacture crankshaft.** *High horsepower* and *economy* are subtypes of *manufacture 8-cylinder crankshaft.* Two mutually exclusive versions of *manufacture 8-cylinder crankshaft* are also depicted, *standard size* and *metric size.*

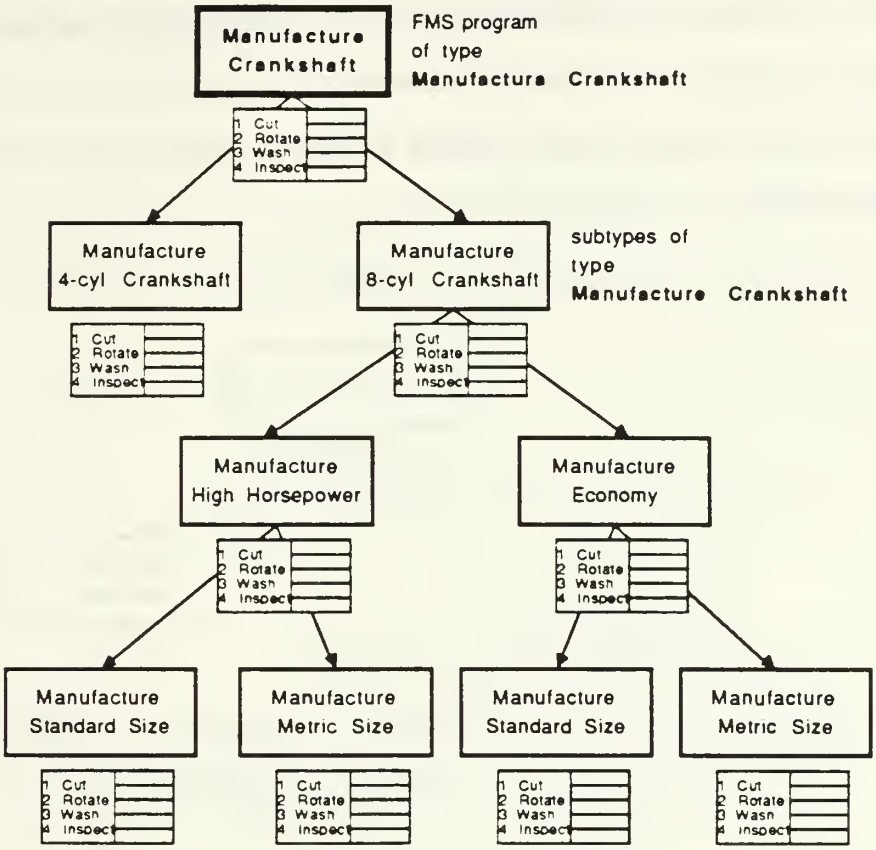


Figure 33. Version Hierarchy of FMS Program **Manufacture Crankshaft**

F. A FMS EXAMPLE

In the section above, we made repeated reference to the Manufacturing Data Model. In this section, we will develop an example by following the steps an industrial engineer would follow in programming an FMS to produce a **crankshaft** using the Manufacturing Data Model. We will assume the engineer must develop the program from scratch.

Initially, the industrial engineer must develop or acquire a top level product drawing to be used as a conceptual schema for the particular piece to be produced. It should be stressed that the engineer needs to be familiar with the product before the programming procedure can begin. In addition, he must be intimately familiar with the capabilities of the FMS in his factory. He can begin the programming process by considering the conceptual schema for the FMS programs he will be using. Figure 34 represents the conceptual schema we will use in this example.

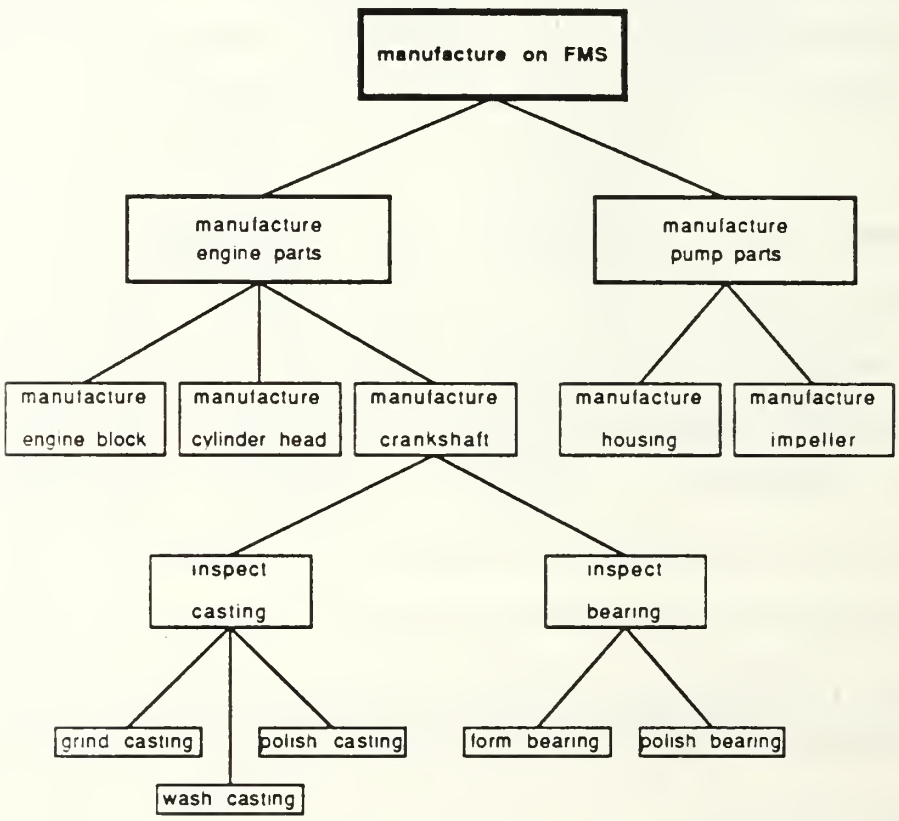


Figure 34. Conceptual Schema of FMS Programs

As the next step, the engineer creates an instance of the FMS program type to be used. Here, we will produce an instance of program type **Manufacture on FMS** namely, *manufacture crankshaft*. Figure 35 depicts this instance.

The third step requires the engineer to identify the component at the lowest level of the schema. In our example, this component would be the subprogram entitled *wash casting*. Beginning with this subprogram, the engineer develops machine programs for the FMS for each primitive level component. For our simple example, he must develop programs for the FMS for the subprograms *grind casting, wash casting, polish casting,*

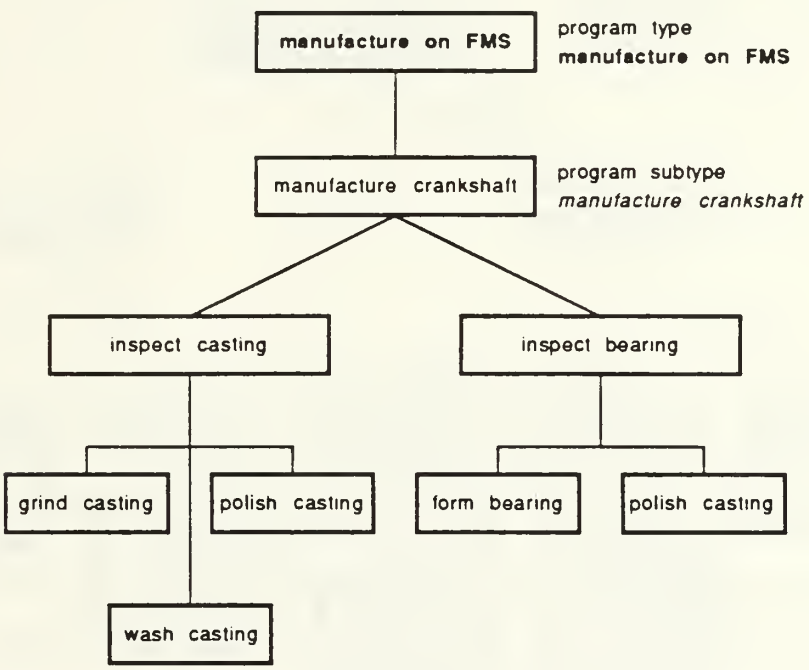


Figure 35. Instance of Program Type **Manufacture on FMS**

form bearing and *polish bearings*. Simplified examples of these subprograms are shown in Figure 36. The engineer must ascertain which information can be specified and which must be parameterized, to be specified later. In our example, all of the subprograms except *grind casting* are fully specified. Let us assume that the Design Department is considering implementation of a revised design. In this case, the *grind casting* program is left parameterized, awaiting further information from Design. This circumstance is depicted in Figure 37. Following the development of these programs, the engineer

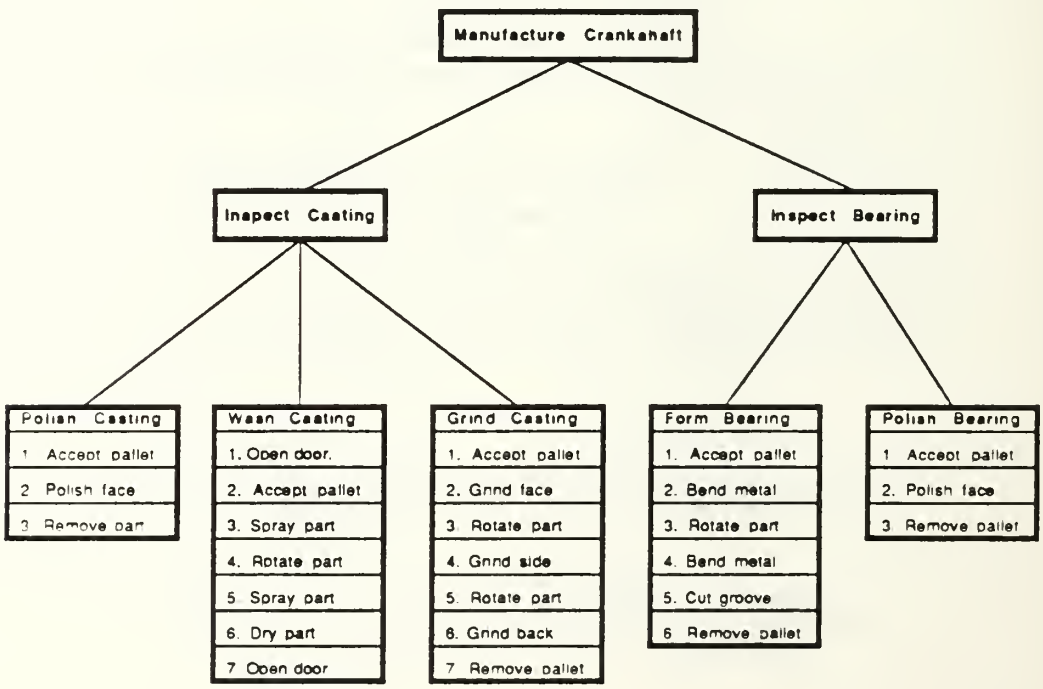


Figure 36. Simplified Examples of Subprograms

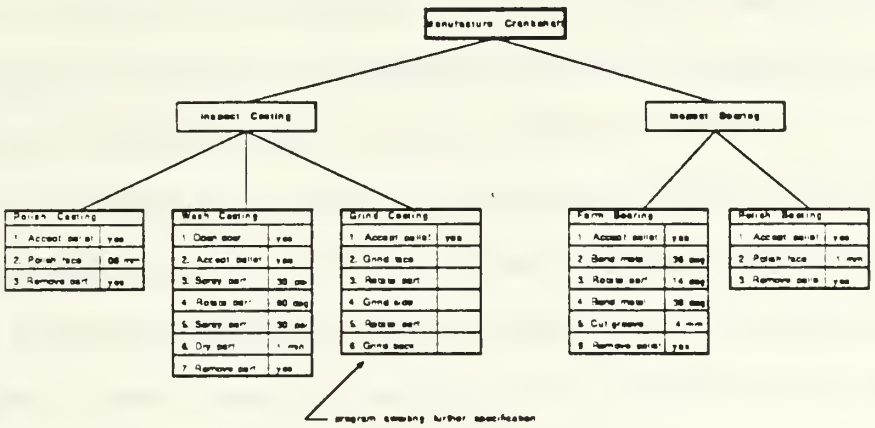


Figure 37. Primitive Level Programs

considers the next level up in the schema.

The FMS program for the next higher level in our example entails inspection of the components. Following inspection, assembly is performed. The *bearing* is fitted to the *crankshaft*, making a crankshaft assembly. Both inspection and assembly require FMS programs. At this point, the process is complete.

One unique aspect of a FMS is the inter-cell transportation system, the integral system which moves parts and assemblies from one cell to another within a FMS. The

transportation cell is not addressed in this example but rather it is treated as a separate entity from the other cells in the FMS. This is analagous to the process planning situation, where the movement of parts and assemblies, represented by the lines on the graphical representation of the conceptual schema, is not considered in the development of the process plan.

The Manufacturing Data Model gives us the ability to use the same data model in all of the manufacturing functions. With minimal modifications, our simple example above could be modified to model the data needs of design, planning or business applications. This capability is the primary advantage of the Model and stands it apart from the rest of the high level data models. In addition, the Manufacturing Data Model has the ability to archive previously used designs and programs, providing a starting point for future efforts and minimizing any duplication of effort. These advantages give the Manufacturing Data Model the potential for increasing the efficiency and productivity of the computer integrated factory.

G. CONCLUSIONS

In the modern marketplace, where the bottom line is the primary concern and where the bottom line is more and more affected by efficiency and productivity, Computer Integrated Manufacturing has become a viable route to business success. Computer Integrated Manufacturing is, however, a misnomer in that the *integrated* refers to integrating computers into manufacturing, rather than using the capabilities of computers to integrate the various functions of manufacturing. In this thesis, we have referenced previous works which identify three approaches to correcting this situation: the high level approach, the centralized database approach, and the low level approach. In Chapter 2, we identified shortcomings in the high level and centralized database approaches such as high cost, the limitations of current technology and limited gains in productivity and

efficiency. We identified a specific low level approach, implementing the Manufacturing Data Model, which shows great promise in capturing the semantics of the manufacturing environment.

The Manufacturing Data Model has been applied, in previous work, to the shop floor layout, process planning and scheduling functions. In these cases, the model demonstrated a reduction in complexity and increased flexibility through the use of several abstraction concepts. Their unique approach, which is data- oriented rather than process-oriented, provides a solution to a complex manufacturing problem.

Flexible Manufacturing Systems, a manufacturing function within Computer Integrated Manufacturing, is a technology which has attempted to integrate manufacturing functions. While the goal of these systems is to integrate their manufacturing cells, in most cases, they have fallen short and instead function as a group of physically proximate numerically controlled machines.

In this chapter, we applied the Manufacturing Data Model to FMS. The model proved capable of modeling the semantics in a simple example involving the manufacture of a can opener in a FMS. What appeared to be a complex task, integrating a complicated machine such as a FMS with a simple product such as a can opener, nonetheless broke down to a very and understandable task using the model.

The Manufacturing Data Model shows the capacity to integrate all of the manufacturing functions using one data model with no translation. This capacity will lead to the ability to integrate the Business Office with the Design Department, Design with the Manufacturing Shop, a FMS with Design, and so forth. This capacity represents a first step on the path to true, and total, Computer Integrated Manufacturing.

Several areas exist for future research in this topic. One area is the implementation of the Manufacturing Data Model, actually coding a representation of the model and

demonstrating, rather than describing, its effectiveness. Another area would be an investigation of the applicability of the model to the design and business office functions. A third area for future research would be the development of a user-interface on a graphics-based system, such as the IRIS workstation, to allow a generic user to interact with the model.

LIST OF REFERENCES

1. Madison, Dana E., Wilbur, Thomas G., and Wu, C. Thomas, *A Data-Oriented Approach to Integrating Manufacturing Functions*, Technical Report, NPS52-87-024, Naval Postgraduate School, Monterey, CA, June 1987.
2. Madison, Dana E. and Wu, C. Thomas, *A Database Approach to Computer Integrated Manufacturing: Scheduling and Shop Floor Layout*, Technical Report, NPS52-87-047, Naval Postgraduate School, Monterey, CA, November 1987.
3. Bunce, P., "Planning for CIM," *The Production Engineer*, v. 64 , p. 21, February 1985.
4. Kochan, Anne and Cowan, Derek, *Implementing CIM*, p. 3, IFS Publications, Ltd., 1986.
5. Korth, Henry F. and Silberschatz, Abraham, *Database System Concepts*, pp. 403-442, Mc Graw-Hill, 1986.
6. Madison, Dana E., *A Database Approach to Computer Integrated Manufacturing*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA, June 1988.
7. Tsichritzis, D. C. and Lochovsky, F. H., *Data Models*, pp. 1-59, Prentice-Hall, 1982.
8. Langefors, B., "Information Systems Theory," *Information Systems*, v. 2, pp. 207-219, 1977.
9. Abrial, J. R., "Data Semantics," in *Data Base Management*, ed. J. W. Klimbie and K. L. Koffeman, pp. 1-59, North-Holland, 1974.
10. Brodie, Michael L. and Ridjanovic, Dzenan, "On the Design and Specification of Database Transactions," in *On Conceptual Modelling*, ed. Joachim W. Schmidt, pp. 277-312, Springer-Verlag, 1984.
11. "CODASYL Data Base Task Group April 71 Report," *ACM*, 1971.
12. Yao, S. B., *Principles of Database Design, vol 1*, Prentice Hall, 1985.
13. Biller, H. and Neuhold, E. J., "Semantics of Databases: The Semantics of Data Models," *Information Systems*, v. 3, pp. 11-30, 1978.
14. Smith, J. M. and Smith, D. C. P., "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems*, v. 2, pp. 105-133, June 1977.
15. Mylopoulos, J., Bernstein, P. A., and Wong, H. K., "A Language Facility for Designing Database-Intensive Applications," *ACM Transactions on Database Systems*, v. 5, pp. 185-207, 1980.
16. Brodie, M. L., "Association: A Database Abstraction for Semantic Modelling," *2nd International Entity-Relationship Conference*, pp. 577-602, 1981.
17. Batory, D. S. and Kim, Won, "Modeling Concepts for VLSI CAD Objects," *ACM Transactions on Database Systems*, v. 10, pp. 322-346, September 1985.

18. Madison, Dana E. and Wu, C. Thomas, *An Expert System Interface and Data Requirements for the Integrated Product Design and Manufacturing Process*, Technical Report, NPS52-86-013, Naval Postgraduate School, Monterey, CA, June 1986.

19. Su, S. Y. W., "Modeling Integrated Manufacturing Data with SAM*," *IEEE Computer*, pp. 34-49, January 1986.

20. Codd, E. F., "Extending the Database Relational Model to Capture More Meaning," *ACM Transactions on Database Systems*, v. 4, pp. 397-434, December 1979.

21. Mylopoulos, J. and Wong, H. K. T., "Some Features of the TAXIS Data Model," *Proceedings of the 6th International Conference on Very Large Databases*, pp. 399-410, 1980.

22. Ranky, Paul G., *Computer Integrated Manufacturing*, pp. 305-428, Prentice/Hall International, 1986.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information System Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Director, Information Systems (OP-945) Office of the Chief of Naval Operations Navy Department Washington, D.C. 20350-2000	1
4.	Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
5.	Superintendent, Naval Postgraduate School Computer Technology Programs, Code 37 Monterey, California 93943-5000	1
6.	Professor C. Thomas Wu, Code 52Wq Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	2
7.	Mr. and Mrs. David R. Fleischman Sr. 15 Brookdale Drive Williamsville, New York 14221	2

Thesis
F4969 Fleischman
c.1 A data oriented approach to integrating manufacturing functions in Flexible Manufacturing Systems.